Table of Contents

Web/service/annotation.ts – *APP 内关于注释配置、管理和检索等任务的 API 接口*

1. fetchAnnotationConfig()

- **属性:**
  - `appId: string` - 应用程序的 ID。
- **返回值:**
  - 描述: 返回 Promise，解析后得到指定应用程序的注释设置。根据应用 ID 获取注释设置详情，包括是否启用、分数阈值和嵌入模型配置。
  - **数据类型:**
    - `id: string` - 注释设置 ID（如果启用）。
    - `enabled: boolean` - 注释功能是否启用。
    - `score_threshold: number` - 启用注释功能所需的分数阈值（如果启用）。
    - `embedding_model: object` - 嵌入模型配置（如果启用），包含以下字段：
      - `embedding_provider_name: string` - 嵌入模型提供者的名称。
      - `embedding_model_name: string` - 嵌入模型的名称。
    - 如果未启用注释功能，返回 `{ enabled: false }`。
- **后端交互:**
  - URL: `apps/${appId}/annotation-setting`
  - 方法: `GET`

2. updateAnnotationStatus()

- **属性:**
  - `appId: string` - 应用程序的 ID。
  - `action: AnnotationEnableStatus` - 启用或禁用操作。
  - `embeddingModel?: EmbeddingModelConfig` - 可选，嵌入模型配置。
  - `score?: number` - 可选，分数阈值。
- **返回值:**
  - 描述: 返回 Promise，解析后进行注释状态更新。根据应用 ID 和操作类型（启用或禁用）更新注释状态，并返回任务 ID 和任务状态。
  - **数据类型:**
    - `job_id: string` - 异步任务的唯一标识符。
    - `job_status: string` - 任务状态，可能的值包括 `waiting`, `processing`。

- **后端交互:**
  - URL: `apps/${appId}/annotation-reply/${action}`
  - 方法: `POST`

3. updateAnnotationScore()

- **属性:**
  - `appId: string` - 应用程序的 ID。
  - `settingId: string` - 注释设置 ID。
  - `score: number` - 分数阈值。
- **返回值:**
  - 描述: 返回 Promise，解析后更新注释分数阈值。根据应用 ID 和设置 ID 更新注释的分数阈值，并返回更新后的注释设置详情。
  - **数据类型:**
    - `id: string` - 注释设置 ID。
    - `enabled: boolean` - 注释功能是否启用。
    - `score_threshold: number` - 更新后的分数阈值。
    - `embedding_model: object` - 嵌入模型配置（如果启用），包含以下字段:
      - `embedding_provider_name: string` - 嵌入模型提供者的名称。
      - `embedding_model_name: string` - 嵌入模型的名称。
- **后端交互:**
  - URL: `apps/${appId}/annotation-settings/${settingId}`
  - 方法: `POST`

4. queryAnnotationJobStatus()

- **属性:**
  - `appId: string` - 应用程序的 ID。
  - `action: AnnotationEnableStatus` - 操作类型（启用或禁用）。
  - `jobId: string` - 任务 ID。
- **返回值:**
  - 描述: 返回 Promise，解析后获取注释任务状态。根据任务 ID 获取任务的当前状态和错误消息（如果有）。
  - **数据类型:**
    - `job_id: string` - 任务 ID。
    - `job_status: string` - 任务状态，可能的值包括 `waiting`, `processing`, `completed`, `failed`，具体值需在 Redis 缓存中检查。
    - `error_msg: string` - 错误消息（如果任务状态是 `failed`）。

- **后端交互:**
  - URL: `apps/${appId}/annotation-reply/${action}/status/${jobId}`
  - 方法: `GET`

5. fetchAnnotationList()

- **属性:**
  - `appId: string` - 应用程序的 ID。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取注释列表。
  - **数据类型:**
    - `data: array` - 注释列表，每个注释对象包含注释 ID 等字段（详细字段需向数据库核对）。
    - `has_more: boolean` - 是否有更多数据。
    - `limit: number` - 每页返回的数据数量。
    - `total: number` - 总数据量。
    - `page: number` - 当前页码。
- **后端交互:**
  - URL: `apps/${appId}/annotations`
  - 方法: `GET`

6. fetchExportAnnotationList()

- **属性:**
  - `appId: string` - 应用程序的 ID。
- **返回值:**
  - 描述: 返回 Promise，解析后导出注释列表。根据应用 ID 获取导出的注释列表。
  - **数据类型:**
    - `data: array` - 导出的注释列表，每个注释对象包含注释 ID 等字段（详细字段需向数据库核对）。
- **后端交互:**
  - URL: `apps/${appId}/annotations/export`
  - 方法: `GET`

7. addAnnotation()

- **属性:**
  - `appId: string` - 应用程序的 ID。
  - `body: AnnotationItemBasic` - 注释的基本信息。
- **返回值:**
  - 描述: 返回 Promise，解析后添加新的注释。根据应用 ID 和提供的注释信息添加新的注释，并返回添加的注释详情。
  - **数据类型:**
    - `id: string` – 应用 ID。
    - `question: string` - 注释的问题。
    - `content: string` - 注释的答案。
    - `account_id: string` – 当前用户 ID。
- **后端交互:**
  - URL: `apps/${appId}/annotations`
  - 方法: `POST`

8. annotationBatchImport()

- **属性:**
  - `url: string` - 导入请求的 URL。
  - `body: FormData` - 包含文件数据的表单数据。
- **返回值:**
  - 描述: 返回 Promise，解析后进行批量导入操作。根据应用 ID 批量导入注释，并返回任务 ID 和任务状态。
  - **数据类型:**
    - `job_id: string` - 导入任务 ID。
    - `job_status: string` - `waiting`。
- **后端交互:**
  - URL: `{URL}`
    - **用例：** `/apps/${appId}/annotations/batch-import (web/app/components/app/annotation/batch-add-annotation-modal/index.tsx line 72)`
  - 方法: `POST`

9. checkAnnotationBatchImportProgress()

- **属性:**
  - `jobID: string` - 导入任务 ID。
  - `appId: string` - 应用程序的 ID。
- **返回值:**
  - 描述: 返回 Promise，解析后检查批量导入进度。根据任务 ID 获取任务的当前状态和错误消息（如果有）。

    - **数据类型:**
        - `job_id: string` - 导入任务 ID。
        - `job_status: string` - 任务状态，可能的值包括 `waiting`, `processing`, `completed`, `failed`。
        - `error_msg: string` - 错误消息（如果任务状态是 `failed`）。
- **后端交互:**
    - URL: `/apps/${appId}/annotations/batch-import-status/${jobID}`
    - 方法: `GET`

10. editAnnotation()

- **属性:**
    - `appId: string` - 应用程序的 ID。
    - `annotationId: string` - 注释 ID。
    - `body: AnnotationItemBasic` - 更新的注释信息。
- **返回值:**
    - 描述: 返回 Promise，解析后更新注释。根据应用 ID 和注释 ID 更新注释内容，并返回更新后的注释详情。
    - **数据类型:**
        - `id: string` - 应用 ID。
        - `question: string` - 注释的问题。
        - `answer: string` - 注释的答案。
        - `account_id: string` -当前用户 ID。
- **后端交互:**
    - URL: `apps/${appId}/annotations/${annotationId}`
    - 方法: `POST`

11. delAnnotation()

- **属性:**
    - `appId: string` - 应用程序的 ID。
    - `annotationId: string` - 注释 ID。
- **返回值:**
    - 描述: 返回 Promise，解析后删除注释。根据应用 ID 和注释 ID 删除指定的注释，并返回删除结果。
    - **数据类型:**
        - `result: string` - `success`。
- **后端交互:**
    - URL: `apps/${appId}/annotations/${annotationId}`
    - 方法: `DELETE`

12. fetchHitHistoryList()

- **属性:**
  - `appId: string` - 应用程序的 ID。
  - `annotationId: string` - 注释 ID。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取命中历史列表。根据应用 ID 和注释 ID 获取注释的命中历史记录。
  - **数据类型:**
    - `data: array` - 命中历史记录列表，每个列表对象包含 ID 等字段（详细字段需向数据库核对）。
    - `has_more: boolean` - 是否有更多数据。
    - `limit: number` - 每页返回的数据数量。
    - `total: number` - 总数据量。
    - `page: number` - 当前页码。
- **后端交互:**
  - URL: `apps/${appId}/annotations/${annotationId}/hit-histories`
  - 方法: `GET`

Web/service/apps.ts – *APP 实体内的增删查取操作及获取统计信息和配置相关的 API 接口*

1. fetchAppList()

- **属性:**
  - `url: string` - 请求的 App 的 URL。
  - `params?: Record<string, any>` - 可选的查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取现有的应用列表。
  - **数据类型:**
    - `data: App[]` - 应用列表。`App` 类型包含：
      - `id: string` - 应用 ID。
      - `name: string` - 应用名称。
      - `description: string` - 应用描述。
      - `icon: string` - 应用图标。
      - `icon_background: string` - 图标背景色。
      - `mode: AppMode` - 应用模式
        - 可能值为 advanced-chat, agent-chat, chat, completion 或 workflow。
      - `enable_site: boolean` - 是否启用 web 应用。
      - `enable_api: boolean` - 是否启用 web API。
      - `api_rpm: number` - 每分钟 API 请求数，默认 60。
      - `api_rph: number` - 每小时 API 请求数，默认 3600。
      - `is_demo: boolean` - 是否为演示应用。
      - `model_config: ModelConfig` - 模型配置
        - `ModelConfig` 详细见`/web/types/app.ts line 141`。
      - `app_model_config: ModelConfig` - 应用模型配置。
        - `ModelConfig` 详细见`/web/types/app.ts line 141`。
      - `created_at: number` - 创建时间。
      - `site: SiteConfig` - Web 应用配置
        - `SiteConfig` 详细见`/web/types/app.ts line 242`。
      - `api_base_url: string` - API 基础 URL。
      - `tags: Tag[]` - 标签列表。`Tag` 类型包含：
        - `id: string` – 标签 ID
        - `name: string` – 标签名称
        - `type: string` – 标签类型
        - `binding_count: number` – 标签所绑定的次数
    - `has_more: boolean` - 是否有更多数据。
    - `limit: number` - 每页数据量限制。
    - `page: number` - 当前页码。

        - `total: number` - 总数据量。
- **后端交互:**
    - URL: `{url}`
        - 用例：`/apps` `(web/context/app-context.tsx line 82)`
    - 方法: `GET`

2. fetchAppDetail()

- **属性:**
    - `url: string` - 请求的 App 的 URL。
    - `id: string` - 应用的 ID。
- **返回值:**
    - 描述: 返回 Promise，解析后获取应用详情。
    - **数据类型:** `App`，`App` 类型包含：
        - `id: string` - 应用 ID。
        - `name: string` - 应用名称。
        - `description: string` - 应用描述。
        - `icon: string` - 应用图标。
        - `icon_background: string` - 图标背景色。
        - `mode: AppMode` - 应用模式。可能值为：
            - advanced-chat, agent-chat, chat, completion 或 workflow。
        - `enable_site: boolean` - 是否启用 web 应用。
        - `enable_api: boolean` - 是否启用 web API。
        - `api_rpm: number` - 每分钟 API 请求数，默认 60。
        - `api_rph: number` - 每小时 API 请求数，默认 3600。
        - `is_demo: boolean` - 是否为演示应用。
        - `model_config: ModelConfig` - 模型配置。
            - `ModelConfig` 详细见`/web/types/app.ts line 141`。
        - `app_model_config: ModelConfig` - 应用模型配置。
            - `ModelConfig` 详细见`/web/types/app.ts line 141`。
        - `created_at: number` - 创建时间。
        - `site: SiteConfig` - Web 应用配置。
            - `SiteConfig` 详细见`/web/types/app.ts line 242`。
        - `api_base_url: string` - API 基础 URL。
        - `tags: Tag[]` - 标签列表。`Tag` 类型包含：
            - `id: string` – 标签 ID
            - `name: string` – 标签名称
            - `type: string` – 标签类型
            - `binding_count: number` – 标签所绑定的次数

- **后端交互:**
  - URL: `{url}/{id}`
    - 用例: `{ url: '/apps', id: appId }` `(web/app/(commonLayout)/app/(appDetailLayout)/[appId]/layout.tsx line 108)`
  - 方法: `GET`

3. fetchAppTemplates()

- **属性:**
  - `url: string` - 请求的 App 的 URL。
- **返回值:**
  - 描述: 返回 Promise，解析后获取应用模板列表。
  - **数据类型:**
    - `data: AppTemplate[]` - 应用模板列表。
      - `name: string` - 模板名称。
      - `description: string` - 模板描述。
      - `mode: AppMode` - 模板模式。可能值为:
        - advanced-chat, agent-chat, chat, completion 或 workflow。
      - `model_config: ModelConfig` - 模型配置。
        - `ModelConfig` 详细见`/web/types/app.ts line 141`。
- **后端交互:**
  - URL: `{url}`
  - 方法: `GET`

4. createApp()

- **属性:**
  - `name: string` - 应用名称。
  - `icon: string` - 应用图标。
  - `icon_background: string` - 图标背景色。
  - `mode: AppMode` - 应用模式。
  - `description?: string` - 可选，应用描述。
  - `config?: ModelConfig` - 可选，模型配置。
- **返回值:**
  - 描述: 返回 Promise，解析后创建新的 App 并获取新创建 App 的详情。
  - **数据类型:** `App`，`App` 类型包含:
    - `id: string` - 应用 ID。
    - `name: string` - 应用名称。
    - `description: string` - 应用描述。
    - `icon: string` - 应用图标。
    - `icon_background: string` - 图标背景色。
    - `mode: AppMode` - 应用模式。可能值为:

        - advanced-chat, agent-chat, chat, completion 或 workflow。
    - `enable_site: boolean` - 是否启用 web 应用。
    - `enable_api: boolean` - 是否启用 web API。
    - `api_rpm: number` - 每分钟 API 请求数，默认 60。
    - `api_rph: number` - 每小时 API 请求数，默认 3600。
    - `is_demo: boolean` - 是否为演示应用。
    - `model_config: ModelConfig` - 模型配置。
        - `ModelConfig` 详细见`/web/types/app.ts line 141`。
    - `app_model_config: ModelConfig` - 应用模型配置。
        - `ModelConfig` 详细见`/web/types/app.ts line 141`。
    - `created_at: number` - 创建时间。
    - `site: SiteConfig` - Web 应用配置。
        - `SiteConfig` 详细见`/web/types/app.ts line 242`。
    - `api_base_url: string` - API 基础 URL。
    - `tags: Tag[]` - 标签列表。`Tag` 类型包含：
        - `id: string` – 标签 ID
        - `name: string` – 标签名称
        - `type: string` – 标签类型
        - `binding_count: number` – 标签所绑定的次数

- **后端交互:**
    - URL: `apps`
    - 方法: `POST`

5. updateAppInfo()

- **属性:**
    - appID: string – 要更新的应用 ID。
    - name: string – 新应用名称。
    - icon: string – 新应用图标。
    - icon_background: string – 新图标背景色。
    - description: string – 新应用描述。
- **返回值:**
    - 描述: 返回 Promise，解析后更新应用信息并获取更新后的 App 实例
    - **数据类型:** `App`
        - 详见 `4.createApp()`返回值
- **后端交互:**
    - URL: `apps/{appID}`
    - 方法: `PUT`

6. copyApp()

- **属性:**
  - `appID: string` - 应用 ID。
  - `name: string` - 新应用名称。
  - `icon: string` - 新应用图标。
  - `icon_background: string` - 新图标背景色。
  - `mode: AppMode` - 应用模式。
  - `description?: string` - 可选，应用描述。
- **返回值:**
  - 描述: 返回 Promise，解析后复制应用并获取所复制的 App 实例。
  - **数据类型:** `App`
    - 详见 `4.createApp()`返回值
- **后端交互:**
  - URL: `apps/{appID}/copy`
  - 方法: `POST`

7. exportAppConfig()

- **属性:**
  - `appID: string` - 应用 ID。
- **返回值:**
  - 描述: 返回 Promise，解析后导出所选 appID 应用配置。
  - **数据类型:**
    - `data: string` - 导出的应用配置数据（YAML format）。
- **后端交互:**
  - URL: `apps/{appID}/export`
  - 方法: `GET`

8. importApp()

- **属性:**
  - `data: string` - 导入的数据。
  - `name?: string` - 可选，新应用名称。
  - `description?: string` - 可选，新应用描述。
  - `icon?: string` - 可选，新应用图标。
  - `icon_background?: string` - 可选，新图标背景色。
- **返回值:**
  - 描述: 返回 Promise，解析后导入新的 App 并获取新倒入 App 的详情。
  - **数据类型:** `App`
- **后端交互:**
  - URL: `apps/import`
  - 方法: `POST`

9. switchApp() – Convert other mode of apps to workflow mode

- **属性:**
  - `appID: string` - 应用 ID。
  - `name: string` - 新应用名称。
  - `icon: string` - 新应用图标。
  - `icon_background: string` - 新图标背景色。
- **返回值:**
  - 描述: 返回 Promise，解析后转换应用并获取新的应用 ID。
  - **数据类型:**
    - `new_app_id: string` - 新应用 ID。
- **后端交互:**
  - URL: `apps/{appID}/convert-to-workflow`
  - 方法: `POST`

10. deleteApp()

- **属性:**
  - `appID: string` – 要删除的应用 ID。
- **返回值:**
  - 描述: 返回 Promise，解析后删除应用并获得删除结果。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 操作结果（`string`）。
- **后端交互:**
  - URL: `apps/{appID}`
  - 方法: `DELETE`

11. updateAppSiteStatus()

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体。
- **返回值:**
  - 描述: 返回 Promise，解析后更新应用站点状态并获得更新后的应用详情。
  - **数据类型:**
    - `App`
      - 详见 `4.createApp()`返回值
- **后端交互:**
  - URL: `{url}`
  - 方法: `POST`

12. updateAppApiStatus()

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体。
- **返回值:**
  - 描述: 返回 Promise，解析后更新应用 API 状态并获得更新后的应用详情。
  - **数据类型：** `App`
    - 详见 `4.createApp()`返回值
- **后端交互:**
  - URL: `{url}`
  - 方法: `POST`

13. updateAppRateLimit()

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体。
- **返回值:**
  - 描述: 返回 Promise，解析后更新应用的速率限制并获得更新后应用详情。
  - **数据类型：** `App`
    - 详见 `4.createApp()`返回值
- **后端交互:**
  - URL: `{url}`
  - 方法: `POST`

14. updateAppSiteAccessToken()

- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - 描述: 返回 Promise，解析后更新应用站点访问令牌并获取更改的应用 ID 以及新的站点设置详情。
  - **数据类型:**
    - `app_id: string` - 应用 ID。
    - `access_token: string` - 应用 URL 标识。
    - `title: string` - 公共标题。
    - `description: string` - 应用描述。
    - `author: string` - 作者。
    - `support_email: string` - 用户支持邮箱地址。
    - `default_language: Language` - 默认语言。
    - `customize_domain: string` - 自定义域名。
    - `theme: string` - 主题。

        - `customize_token_strategy: 'must' | 'allow' | 'not_allow'` - 自定义 Token 策略用户是否能选择自己的 openAI Key。
        - `prompt_public: boolean` - 提示是否公开。
        - `app_base_url: string` - Web API 和 APP 基础域名。
        - `copyright: string` - 版权信息。
        - `privacy_policy: string` - 隐私政策。
        - `custom_disclaimer: string` - 自定义免责声明。
        - `icon: string` - 图标。
        - `icon_background: string` - 图标背景色。
        - `show_workflow_steps: boolean` - 显示工作流程步骤。
- **后端交互:**
    - URL: `{url}`
        - 用例：`/apps/${appId}/site/access-token-reset (web/app/(commonLayout)/app/(appDetailLayout)/[appId]/overview/cardView.tsx line 90)`
    - 方法: `POST`

15. updateAppSiteConfig()

- **属性:**
    - `url: string` - 请求的 URL。
    - `body: Record<string, any>` - 请求体。
- **返回值:**
    - 描述: 返回 Promise，解析后更新应用站点配置并获取更新后站点的详情。
    - **数据类型:**
        - `App`, 详见 `4.createApp()`返回值
- **后端交互:**
    - URL: `{url}`
        - 用例：`/apps/${appId}/site (web/app/(commonLayout)/app/(appDetailLayout)/[appId]/overview/cardView.tsx line 77)`
    - 方法: `POST`

16. getAppDailyConversations()

- **属性:**
    - `url: string` - 请求的 URL。
    - `params: Record<string, any>` - 查询参数。
- **返回值:**
    - 描述: 返回 Promise，解析后获取应用每日对话数量纪录。
    - **数据类型:**
        - `data: Array<{ date: string; conversation_count: number }>` - 每日对话记录数据。

- **后端交互:**
  - URL: `{url}`
    - 用例: `/apps/${id}/statistics/daily-conversations (web/app/components/app/overview/appChart.tsx line 263)`
  - 方法: `GET`

17. getWorkflowDailyConversations()

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取工作流每日对话数量记录。
  - **数据类型:**
    - `data: Array<{ date: string; runs: number }>` - 每日工作流对话记录数据。
- **后端交互:**
  - URL: `{url}`
    - 用例: `/apps/${id}/workflow/statistics/daily-conversations (web/app/components/app/overview/appChart.tsx line 374)`
  - 方法: `GET`

18. getAppStatistics

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取应用统计数据。
  - **数据类型:**
    - `data: Array<{ date: string }>` - 应用统计数据。
- **后端交互:**
  - URL: `{url}`
    - 用例: `/apps/${id}/statistics/… (web/app/components/app/overview/appChart.tsx line 292/308/325/342/419)`
  - 方法: `GET`

19. getAppDailyEndUsers

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。

- **返回值:**
  - 描述: 返回 Promise，解析后获取应用每日终端用户数量数据。
  - **数据类型:**
    - `data: Array<{ date: string; terminal_count: number }>` - 每日终端用户数据。
- **后端交互:**
  - URL: `{url}`
    - 用例：详见 `(web/app/components/app/overview/appChart.tsx line 278/390)`
  - 方法: `GET`

20. getAppTokenCosts

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取应用 Token 消耗数据。
  - **数据类型:**
    - `data: Array<{`

      `date: string;`

      `token_count: number;`

      `total_price: number;`

      `currency: number }>` - 每日 Token 消耗数据。

- **后端交互:**
  - URL: `{url}`
    - 用例：`/apps/${id}/(workflow/)statistics/token-costs` `(web/app/components/app/overview/appChart.tsx line 360/405)`
  - 方法: `GET`

21. updateAppModelConfig

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体。
- **返回值:**
  - 描述: 返回 Promise，解析后更新应用模型配置。
  - **数据类型:**
    - `result: string` - 更新结果。

- **后端交互:**
  - URL: `{url}`
    - 用例：`/apps/${appId}/model-config (web/app/components/app/configuration/index.tsx line 627)`
  - 方法: `POST`

22. fetchAppListNoMock

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取应用列表（无 Mock）。
  - **数据类型:**
    - `data: App[]` - 应用列表。
    - `has_more: boolean` - 是否有更多数据。
    - `limit: number` - 每页数据量限制。
    - `page: number` - 当前页码。
    - `total: number` - 总数据量。
- **后端交互:**
  - URL: `{url}`
  - 方法: `GET`

23. fetchApiKeysList

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后获取 API Key 列表。
  - **数据类型:**
    - `data: ApikeyItemResponse[]` - API Key 列表。
      - `id: string` - API Key ID。
      - `token: string` - API Key Token。
      - `last_used_at: string` - 上次使用时间。
      - `created_at: string` - 创建时间。
- **后端交互:**
  - URL: `{url}`
    - 用例：`/apps/${appId}/api-keys (web/app/components/develop/secret-key/secret-key-modal.tsx line 52 -> 50)`
  - 方法: `GET`

24. delApikey

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 查询参数。
- **返回值:**
  - 描述: 返回 Promise，解析后删除 API Key 并返回删除结果。
  - **数据类型:**
    - `result: 'success' | 'fail'` - 操作结果。
- **后端交互:**
  - URL: `{url}`
    - 用例：`/apps/${appId}/api-keys/${delKeyID}` (web/app/components/develop/secret-key/secret-key-modal.tsx line 76 -> 78)
  - 方法: `DELETE`

25. createApikey

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体。
- **返回值:**
  - 描述: 返回 Promise，解析后创建 API Key 并获取新创建 Key 的详情。
  - **数据类型:**
    - `id: string` - API Key ID。
    - `token: string` - API Key Token。
    - `created_at: string` - 创建时间。
- **后端交互:**
  - URL: `{url}`
    - 用例：`/apps/${appId}/api-keys` (web/app/components/develop/secret-key/secret-key-modal.tsx line 88 -> 86)
  - 方法: `POST`

26. validateOpenAIKey

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: { token: string }` - 请求体。
- **返回值:**
  - 描述: 返回 Promise，解析后验证 OpenAI Key 并获取验证结果。
  - **数据类型:**
    - `result: string` - 验证结果。

        - `error?: string` - 错误信息（如果有）。
- **后端交互:**
    - URL: `{url}`
    - 方法: `POST`

27. updateOpenAIKey

- **属性:**
    - `url: string` - 请求的 URL。
    - `body: { token: string }` - 请求体。
- **返回值:**
    - 描述: 返回 Promise，解析后更新 OpenAI Key 并获取更新结果。
    - **数据类型:**
        - `result: string` - 更新结果。
        - `error?: string` - 错误信息（如果有）。
- **后端交互:**
    - URL: `{url}`
    - 方法: `POST`

28. generationIntroduction

- **属性:**
    - `url: string` - 请求的 URL。
    - `body: { prompt_template: string }` - 请求体。
- **返回值:**
    - 描述: 返回 Promise，解析后生成并获得引导词。
    - **数据类型:**
        - `introduction: string` - 生成的引导词。
- **后端交互:**
    - URL: `{url}`
    - 方法: `POST`

29. fetchAppVoices

- **属性:**
    - `appId: string` - 应用 ID。
    - `language?: string` - 可选，语言。
- **返回值:**
    - 描述: 返回 Promise，解析后获取应用的语音列表。
    - **数据类型:**

```
[{
```

        - `name: string` - 语音名称。
        - `value: string` - 语音值。

    `}]`

- **后端交互:**
    - URL: `apps/${appId}/text-to-audio/voices?language=${language}`
    - 方法: `GET`

Web/service/base.ts – *APP 主体核心功能，包括对流数据、身份验证、错误处理支持等的 API 接口*

1. unicodeToChar

- **用途:** 将 Unicode 数据转换为对应的字符
- **属性:**
    - `text: string` - 要转换的 Unicode 字符串。
- **返回值:**
    - **描述:** 返回转换后的普通字符串。
    - **数据类型:** `string`

2. requiredWebSSOLogin

- **用途:** 当需要 Web SSO 登录时，重定向用户到指定的登录页面。
- **属性:** 无
- **返回值:**
    - **描述:** 无返回值，但会将用户重定向到 Web SSO 登录页面。
    - **数据类型:** `void`

3. format

- **用途:** 将普通文本格式化为 HTML 格式，替换换行符和代码块标记。
- **属性:**
    - `text: string` - 要格式化的文本。
- **返回值:**
    - **描述:** 返回格式化后的 HTML 文本。
    - **数据类型:** `string`

4. handleStream

- **用途:** 处理服务器发送的事件流，通过 Server-Sent Events (SSE) 协议传递。在前端应用程序中实时接收和处理来自服务器的更新数据。非常适合用于需要持续更新和实时数据的应用场景，如聊天应用、实时监控系统、工作流管理系统等。
- **属性:**
  - `response: Response` - HTTP 响应对象。
  - `onData: IOnData` - 数据处理回调函数。
    - `message: string` - 事件消息内容。
    - `isFirstMessage: boolean` - 是否为第一条消息。
    - `moreInfo: IOnDataMoreInfo` - 其他信息。
      - `conversationId?: string` - 对话 ID（可选）。
      - `taskId?: string` - 任务 ID（可选）。
      - `messageId: string` - 消息 ID。
      - `errorMessage?: string` - 错误消息（可选）。
      - `errorCode?: string` - 错误代码（可选）。
  - `onCompleted?: IOnCompleted` - 可选的完成处理回调函数。
    - `hasError?: boolean` - 是否有错误（可选）。
    - `errorMessage?: string` - 错误消息（可选）。
  - `onThought?: IOnThought` - 可选的思路处理回调函数。
    - `thought: ThoughtItem` - 思路对象。
      - `id: string` - 思路 ID。
      - `text: string` - 思路内容。
      - `timestamp: number` - 时间戳。
  - `onFile?: IOnFile` - 可选的文件处理回调函数。
    - `file: VisionFile` - 文件对象。
      - `id: string` - 文件 ID。
      - `name: string` - 文件名称。
      - `url: string` - 文件 URL。
      - `size: number` - 文件大小。
  - `onMessageEnd?: IOnMessageEnd` - 可选的消息结束处理回调函数。
    - `messageEnd: MessageEnd` - 消息结束对象。
      - `messageId: string` - 消息 ID。
      - `conversationId: string` - 对话 ID。
  - `onMessageReplace?: IOnMessageReplace` - 可选的消息替换处理回调函数。
    - `messageReplace: MessageReplace` - 消息替换对象。
      - `oldMessageId: string` - 旧消息 ID。
      - `newMessageId: string` - 新消息 ID。
      - `newText: string` - 新消息文本。
  - `onWorkflowStarted?: IOnWorkflowStarted` - 可选的工作流开始处理回调函数。

        - `workflowStarted: WorkflowStartedResponse` - 工作流开始对象。
            - `task_id: string` - 任务 ID。
            - `workflow_run_id: string` - 工作流运行 ID。
            - `event: string` - 事件名称。
            - `data: WorkflowStartedData` - 数据对象。
                - `id: string` - 数据 ID。
                - `workflow_id: string` - 工作流 ID。
                - `sequence_number: number` - 序列号。
                - `created_at: number` - 创建时间。
    - `onWorkflowFinished?: IOnWorkflowFinished` - 可选的工作流结束处理回调函数。
        - `workflowFinished: WorkflowFinishedResponse` - 工作流结束对象。
            - `task_id: string` - 任务 ID。
            - `workflow_run_id: string` - 工作流运行 ID。
            - `event: string` - 事件名称。
            - `data: WorkflowFinishedData` - 数据对象。
                - `id: string` - 数据 ID。
                - `workflow_id: string` - 工作流 ID。
                - `status: string` - 状态。
                - `outputs: any` - 输出数据。
                - `error: string` - 错误信息。
                - `elapsed_time: number` - 运行时间。
                - `total_tokens: number` - 总 Token 数。
                - `total_steps: number` - 总步骤数。
                - `created_at: number` - 创建时间。
                - `created_by: CreatedBy` - 创建人信息。
                    - `id: string` - 创建人 ID。
                    - `name: string` - 创建人姓名。
                    - `email: string` - 创建人邮箱。
                - `finished_at: number` - 结束时间。
    - `onNodeStarted?: IOnNodeStarted` - 可选的节点开始处理回调函数。
        - `nodeStarted: NodeStartedResponse` - 节点开始对象。
            - `task_id: string` - 任务 ID。
            - `workflow_run_id: string` - 工作流运行 ID。
            - `event: string` - 事件名称。
            - `data: NodeStartedData` - 数据对象。
                - `id: string` - 数据 ID。
                - `node_id: string` - 节点 ID。
                - `node_type: string` - 节点类型。
                - `index: number` - 索引。

            - `predecessor_node_id?: string` - 前置节点 ID（可选）。
            - `inputs: any` - 输入数据。
            - `created_at: number` - 创建时间。
            - `extras?: any` - 额外信息（可选）。
    - `onNodeFinished?: IOnNodeFinished` - 可选的节点结束处理回调函数。
        - `nodeFinished: NodeFinishedResponse` - 节点结束对象。
            - `task_id: string` - 任务 ID。
            - `workflow_run_id: string` - 工作流运行 ID。
            - `event: string` - 事件名称。
            - `data: NodeFinishedData` - 数据对象。
                - `id: string` - 数据 ID。
                - `node_id: string` - 节点 ID。
                - `node_type: string` - 节点类型。
                - `index: number` - 索引。
                - `predecessor_node_id?: string` - 前置节点 ID（可选）。
                - `inputs: any` - 输入数据。
                - `process_data: any` - 处理数据。
                - `outputs: any` - 输出数据。
                - `status: string` - 状态。
                - `error: string` - 错误信息。
                - `elapsed_time: number` - 运行时间。
                - `execution_metadata: ExecutionMetadata` - 执行元数据。
                    - `total_tokens: number` - 总 Token 数。
                    - `total_price: number` - 总价格。
                    - `currency: string` - 货币。
                - `created_at: number` - 创建时间。
    - `onIterationStart?: IOnIterationStarted` - 可选的迭代开始处理回调函数。
        - `iterationStarted: IterationStartedResponse` - 迭代开始对象。
            - `task_id: string` - 任务 ID。
            - `workflow_run_id: string` - 工作流运行 ID。
            - `event: string` - 事件名称。
            - `data: IterationStartedData` - 数据对象。
                - `id: string` - 数据 ID。
                - `node_id: string` - 节点 ID。
                - `metadata: IterationMetadata` - 元数据。
                    - `iterator_length: number` - 迭代长度。
                - `created_at: number` - 创建时间。

        - extras?: any - 额外信息（可选）。
    - onIterationNext?: IOnIterationNexted - 可选的迭代下一个处理回调函数。
        - iterationNexted: IterationNextedResponse - 迭代下一个对象。
            - task_id: string - 任务 ID。
            - workflow_run_id: string - 工作流运行 ID。
            - event: string - 事件名称。
            - data: IterationNextedData - 数据对象。
                - id: string - 数据 ID。
                - node_id: string - 节点 ID。
                - index: number - 索引。
                - output: any - 输出数据。
                - extras?: any - 额外信息（可选）。
                - created_at: number - 创建时间。
    - onIterationFinish?: IOnIterationFinished - 可选的迭代结束处理回调函数。
        - iterationFinished: IterationFinishedResponse - 迭代结束对象。
            - task_id: string - 任务 ID。
            - workflow_run_id: string - 工作流运行 ID。
            - event: string - 事件名称。
            - data: IterationFinishedData - 数据对象。
                - id: string - 数据 ID。
                - node_id: string - 节点 ID。
                - outputs: any - 输出数据。
                - extras?: any - 额外信息（可选）。
                - status: string - 状态。
                - created_at: number - 创建时间。
                - error: string - 错误信息。
    - onTextChunk?: IOnTextChunk - 可选的文本块处理回调函数。
        - textChunk: TextChunkResponse - 文本块对象。
            - task_id: string - 任务 ID。
            - workflow_run_id: string - 工作流运行 ID。
            - event: string - 事件名称。
            - data: TextChunkData - 数据对象。
                - text: string - 文本内容。
    - onTextReplace?: IOnTextReplace - 可选的文本替换处理回调函数。
        - textReplace: TextReplaceResponse - 文本替换对象。
            - task_id: string - 任务 ID。
            - workflow_run_id: string - 工作流运行 ID。
            - event: string - 事件名称。
            - data: TextReplaceData - 数据对象。

        - `text: string` - 文本内容。
- **返回值:**
    - **描述:** 无返回值，通过调用回调函数来输出处理结果。具体输出如下：
        - `onData`: 每当接收到新的消息数据时调用，输出消息内容及相关元信息。
        - `onCompleted`: 数据流处理完毕时调用，输出处理是否出错及错误信息。
        - `onThought`: 每当接收到新的思路数据时调用，输出思路内容。
        - `onFile`: 每当接收到新的文件数据时调用，输出文件信息。
        - `onMessageEnd`: 每当一条消息处理完毕时调用，输出消息结束信息。
        - `onMessageReplace`: 每当接收到消息替换事件时调用，输出替换后的新消息信息。
        - `onWorkflowStarted`: 工作流开始时调用，输出工作流开始信息。
        - `onWorkflowFinished`: 工作流结束时调用，输出工作流结束信息。
        - `onNodeStarted`: 节点开始时调用，输出节点开始信息。
        - `onNodeFinished`: 节点结束时调用，输出节点结束信息。
        - `onIterationStart`: 迭代开始时调用，输出迭代开始信息。
        - `onIterationNext`: 迭代进行下一步时调用，输出迭代下一步信息。
        - `onIterationFinish`: 迭代结束时调用，输出迭代结束信息。
        - `onTextChunk`: 接收到文本块数据时调用，输出文本内容。
        - `onTextReplace`: 接收到文本替换数据时调用，输出替换后的文本内容。
    - **数据类型:** `void`

5. baseFetch

- **用途:** 通用的 HTTP 请求函数（ `GET`、`POST`、`PUT`、`DELETE` 等），支持发送请求，处理认证、处理请求参数、设置请求头、处理超时等功能。
- **属性:**
    - `url: string` - 请求的 URL。
    - `fetchOptions: FetchOptionType` - Fetch 请求选项。
        - `FetchOptionType` 详细见 `web/service/base.ts line 96`。
    - `otherOptions: IOtherOptions` - 其他选项。
        - `IOtherOptions` 详细见 `web/service/base.ts line 64`。
- **返回值:**
    - **描述:** 返回 Promise，用于处理 HTTP 请求，解析后提供响应数据：
        - **成功响应:**
            - 返回解析后的响应数据（通常为 JSON 格式）。
            - 根据 `method` 的不同，实现不同的请求类型（`GET`、`POST`、`PUT`、`DELETE` 等）；根据 `Content-Type`（`base.ts line 32 -> 19`）的不同，可能返回 JSON 数据或 Blob（二进制数据，如文件）。

        - **错误响应:**
            - 如果请求失败或返回错误状态码，Promise 将被拒绝，并抛出异常。
            - 错误处理包括显示错误提示（如使用 `Toast` 组件）和重新认证（如处理 401 未授权状态）。
    - **数据类型:** `Promise<T>`
- **后端交互:**
    - **URL:** `{url}`。
    - **方法:** 根据 `fetchOptions` 动态设置。

6. upload

- **用途:** 处理文件上传请求，使用 `XMLHttpRequest` 对象发送文件数据到服务器，并支持进度事件的处理。通过监听 `onprogress` 事件，可以实时获取文件上传的进度。
- **属性:** (范例见 `web/app/components/datasets/create/file-uploader/index.tsx line 115`)
    - `options: any` - 上传选项。
    - `isPublicAPI?: boolean` - 是否为公共 API。
    - `url?: string` - 可选的 URL。
    - `searchParams?: string` - 可选的查询参数。
- **返回值:**
    - **描述:** 返回 Promise，用于处理文件上传，解析后提供上传结果。具体输出如下：
        - **成功响应:**
            - 返回上传成功后的响应数据，包含上传文件的相关信息。
        - **错误响应:**
            - 抛出异常。
        - **进度事件:**
            - 通过 `onprogress` 事件处理器（`web/app/components/datasets/create/file-uploader/index.tsx line 108`），可以获取文件上传进度的实时更新。
    - **数据类型:** `Promise<any>`
- **后端交互:**
    - **URL:** 基于参数 `url` 和 `isPublicAPI`。
    - **方法:** POST

7. ssePost

- **用途:** 用于发送 POST 请求并处理服务器发送的事件流 (SSE)。适合需要从服务器接收实时更新的场景，如实时通知、数据流处理等。通过 `ssePost`，前端可以与服务器保持一个持久的连接，接收并处理服务器发送的实时数据。
- **属性:**
  - `url: string` - 请求的 URL。
  - `fetchOptions: FetchOptionType` - Fetch 请求选项。
    - 详见 `web/service/base.ts line 96`
  - `otherOptions: IOtherOptions` - 其他选项。
    - 详见 `web/service/base.ts line 64`
- **返回值:**
  - **描述:** 无返回值，通过调用回调函数来处理服务器发送的事件流（SSE），具体调用如下：
    - `onData`:每当接收到新的消息数据时调用，输出消息内容及相关元信息。
    - 详见 handleStream()返回值
  - **数据类型:** `void`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** POST

8. request

- **用途:** 基础请求方法，用于发送任意 HTTP 请求。通过结合 `baseFetch()` 函数的功能，提供了统一的接口来处理 `GET`、`POST`、`PUT`、`DELETE` 等请求方法。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项（范例见 `web/service/base.ts line 545`）。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，用于处理 HTTP 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`。
  - **方法:** 动态设置，基于 `options`。

9. **get**

- **用途:** 发送 GET 请求以获取数据。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，用于处理 GET 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** GET

10. **getPublic**

- **用途:**专门处理需要公共 API 前缀和认证的 GET 请求。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，调用 `get` 方法并设置 `isPublicAPI` 为 `true`，用于处理公共 API 的 `GET` 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** GET

11. **post**

- **用途:** 发送 POST 请求以提交数据。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，用于处理 POST 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** POST

12. **postPublic**

- **用途:** 发送公共 API 的 POST 请求。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，调用 `post` 方法并设置 `isPublicAPI` 为 `true`，用于处理公共 API 的 POST 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** POST

13. **put**

- **用途:** 发送 PUT 请求以更新数据。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，用于处理 PUT 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** PUT

14. **putPublic**

- **用途:** 发送公共 API 的 PUT 请求。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:** 返回 Promise，调用 `put` 方法并设置 `isPublicAPI` 为 `true`，用于处理公共 API 的 PUT 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`

- **方法:** PUT

15. del

- **用途:** 发送 DELETE 请求以删除数据。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:**返回 Promise，用于处理 DELETE 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** DELETE

16. delPublic

- **用途:** 发送公共 API 的 DELETE 请求。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:**返回 Promise，调用 `del` 方法并设置 `isPublicAPI` 为 `true`，用于处理公共 API 的 DELETE 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** DELETE

17. patch

- **用途:** 发送 PATCH 请求以更新服务器上的部分资源。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:**返回 Promise，用于处理 PATCH 请求。
  - **数据类型:** `Promise<T>`

- **后端交互:**
  - **URL:** `{url}`
  - **方法:** PATCH

18. patchPublic

- **用途:** 发送公共 API 的 PATCH 请求。
- **属性:**
  - `url: string` - 请求的 URL。
  - `options?: object` - 请求选项。
  - `otherOptions?: IOtherOptions` - 其他选项。
- **返回值:**
  - **描述:**返回 Promise，调用 `patch` 方法并设置 `isPublicAPI` 为 `true`，用于处理公共 API 的 PATCH 请求。
  - **数据类型:** `Promise<T>`
- **后端交互:**
  - **URL:** `{url}`
  - **方法:** PATCH

Web/service/billing.ts – *从后端获取计费和订阅信息的 API 接口*

1. fetchCurrentPlanInfo()

- **属性:**
  - 无。
- **返回值:**
  - 描述: 返回 Promise，解析后获取当前计划的信息。根据当前租户 ID 获取当前的订阅计划和特性信息。
  - **数据类型:**
    - `billing: object` – 计费信息。
      - `enabled: boolean` – 计费功能是否启用。
      - `subscription: object` – 订阅信息。
        - `plan: string` – 当前订阅的计划类型。
    - `members: object` – 成员信息。
      - `size: number` – 当前成员数量。
      - `limit: number` – 成员数量限制，0 表示无限制。
    - `apps: object` – 应用信息。
      - `size: number` – 当前应用数量。
      - `limit: number` – 应用数量限制，0 表示无限制。
    - `vector_space: object` – 向量空间信息。

        - `size: number` - 当前向量空间大小。
        - `limit: number` - 向量空间大小限制，0 表示无限制。
      - `annotation_quota_limit: object` - 注释配额信息。
        - `size: number` - 当前注释数量。
        - `limit: number` - 注释数量限制，0 表示无限制。
      - `documents_upload_quota: object` - 文档上传配额信息。
        - `size: number` - 当前上传的文档数量。
        - `limit: number` - 上传文档数量限制，0 表示无限制。
      - `docs_processing: string` - 文档处理优先级。
      - `can_replace_logo: boolean` - 是否可以替换标志。
      - `model_load_balancing_enabled: boolean` - 是否启用模型负载均衡。
- **后端交互:**
  - URL: `/features`
  - 方法: `GET`

2. fetchSubscriptionUrls()

- **属性:**
  - `plan: string` - 订阅计划类型。
  - `interval: string` - 订阅周期。
- **返回值:**
  - 描述: 返回 Promise，解析后获取订阅 URL。根据订阅计划类型和订阅周期获取相应的订阅 URL。
  - **数据类型:**
    - `payment_url: string` - 订阅结算页面的 URL。(GET `/subscription/payment-link` 的返回值)
- **后端交互:**
  - URL: `/billing/subscription?plan=${plan}&interval=${interval}`
  - 方法: `GET`

3. fetchBillingUrl()

- **属性:**
  - 无
- **返回值:**
  - 描述: 返回 Promise，解析后获取当前用户的账单 URL。
  - **数据类型:**
    - `url: string` - 账单页面的 URL。
- **后端交互:**
  - URL: `/billing/invoices`
  - 方法: `GET`

Web/service/common.ts – *APP 用户管理、工作区操作、集成设置和系统配置相关的 API 接口*

1. login

- **用途:**用于处理用户登录请求。它通过发送 POST 请求，将用户的登录信息发送到服务器进行验证。成功后，服务器会返回登录结果和相关数据。
- **属性:** (范例见 `web/app/signin/normalForm.tsx line 89`)
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含用户的登录信息，如用户名和密码。
- **返回值:**
  - **描述:** 返回一个 Promise，用于处理登录请求，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse & { data: string }` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
      - `data: string` - 额外的字符串数据。
- **后端交互:**
  - **URL:** {`url`}
    - 用例：`/login` (`web/app/signin/normalForm.tsx line 90`)
  - **方法:** POST

2. setup

- **用途:**处理系统的初始化设置请求。通过发送 POST 请求，向服务器提交初始化设置数据，服务器返回设置结果。
- **属性:**
  - `body: Record<string, any>` - 请求体，包含初始化设置的数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于处理初始化设置请求，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** `/setup`
  - **方法:** POST

3. fetchSetupStatus

- **用途:**获取系统初始化设置的状态。通过发送 GET 请求，从服务器获取当前的设置状态。
- **属性:**
  - 无参数
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取初始化设置状态，解析后提供响应数据。
  - **数据类型:**
    - `SetupStatusResponse` - 响应数据类型。
      - `step: 'finished' | 'not_started'` - 初始化步骤状态。
      - `setup_at?: Date` - 初始化完成时间（可选）。
- **后端交互:**
  - **URL:** `/setup`
  - **方法:** GET

4. initValidate

- **用途:**用于处理系统初始化验证请求。通过发送 POST 请求，向服务器提交验证数据，服务器返回验证结果。
- **属性:**
  - `body: Record<string, any>` - 请求体，包含初始化验证的数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于处理初始化验证请求，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** `/init`
  - **方法:** POST

5. fetchInitValidateStatus

- **用途:** 获取系统初始化验证的状态。通过发送 GET 请求，从服务器获取当前的验证状态。
- **属性:**
  - 无参数
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取初始化验证状态，解析后提供响应数据。

    - **数据类型:**
        - `InitValidateStatusResponse` - 响应数据类型。
            - `status: 'finished' | 'not_started'` - 验证状态。
- **后端交互:**
    - **URL:** `/init`
    - **方法:** GET

6. fetchUserProfile

- **用途:**获取用户的个人资料信息。通过发送 GET 请求，从服务器获取当前用户的详细信息。
- **属性:**
    - `url: string` - 请求的 URL。
    - `params: Record<string, any>` - 请求参数，包含查询用户信息的必要参数。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取用户个人资料，解析后提供响应数据。
    - **数据类型:**
        - `UserProfileOriginResponse` - 响应数据类型。
            - `json: () => Promise<UserProfileResponse>` - 返回解析后的用户信息。
                - `id: string` - 用户 ID。
                - `name: string` - 用户名。
                - `email: string` - 用户邮箱。
                - `avatar: string` - 用户头像 URL。
                - `is_password_set: boolean` - 是否设置了密码。
                - `interface_language?: string` - 界面语言（可选）。
                - `interface_theme?: string` - 界面主题（可选）。
                - `timezone?: string` - 时区（可选）。
                - `last_login_at?: string` - 最后登录时间（可选）。
                - `last_active_at?: string` - 最后活跃时间（可选）。
                - `last_login_ip?: string` - 最后登录 IP（可选）。
                - `created_at?: string` - 创建时间（可选）。
            - `bodyUsed: boolean` - 是否使用了请求体。
            - `headers: any` - 响应头信息。
- **后端交互:**
    - **URL:** {url}(`/account/profile: web/context/app-context.tsx line 83`)
    - **方法:** GET

7. updateUserProfile

- **用途:**用于更新用户的个人资料信息。通过发送 POST 请求，将更新后的用户信息提交到服务器。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含要更新的用户信息。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新用户个人资料，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/account/name: web/app/components/header/account-setting/account-page/index.tsx line 55`)
  - **方法:** POST

8. logout

- **用途:**用于处理用户登出请求。通过发送 GET 请求，将用户的登出信息发送到服务器进行处理。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含用户的注销信息。
- **返回值:**
  - **描述:** 返回一个 Promise，用于处理注销请求，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/logout: web/app/components/header/account-dropdown/index.tsx line 39`)
  - **方法:** GET

9. fetchLanggeniusVersion

- **用途:**获取当前 LangGenius 的版本信息。通过发送 GET 请求，从服务器获取版本的详细信息。

- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含查询版本信息的必要参数。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取版本信息，解析后提供响应数据。
  - **数据类型:**
    - `LangGeniusVersionResponse` - 响应数据类型。
      - `current_version: string` - 当前版本。
      - `latest_version: string` - 最新版本。
      - `version: string` - 版本号。
      - `release_date: string` - 发布日期。
      - `release_notes: string` - 发布说明。
      - `can_auto_update: boolean` - 是否支持自动更新。
      - `current_env: string` - 当前环境。
- **后端交互:**
  - **URL:** {url} (`/version: web/context/app-context.tsx line 98`)
  - **方法:** GET

10. oauth

- **用途:**用于处理 OAuth 验证请求。通过发送 GET 请求，向服务器提交 OAuth 验证数据，并获取验证结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含 OAuth 验证需要的数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于处理 OAuth 验证请求，解析后提供响应数据。
  - **数据类型:**
    - `OauthResponse` - 响应数据类型。
      - `redirect_url: string` - 重定向 URL。
- **后端交互:**
  - **URL:** {url} (`web/app/signin/normalForm.tsx line 115/124`)
  - **方法:** GET

11. oneMoreStep

- **用途:**用于处理 `Sign in` 过程中的额外的步骤请求，专门用于处理邀请码、选择界面语言和选择时区。

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含额外步骤的数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于处理额外步骤请求，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url}（`/account/init: web/app/signin/oneMoreStep.tsx line 60`）
  - **方法:** POST

12. fetchMembers

- **用途:**用于获取团队成员列表。通过发送 GET 请求，从服务器获取当前团队的成员信息。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含查询成员信息的参数。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取团队成员列表，解析后提供响应数据。
  - **数据类型:**
    - `{ accounts: Member[] | null }` - 响应数据类型。
      - `accounts: Member[] | null` - 团队成员列表。（详见 `web/models/common.ts line 65 -> 20`）
        - `id: string` - 用户 ID。
        - `name: string` - 用户名。
        - `email: string` - 用户邮箱。
        - `avatar: string` - 用户头像 URL。
        - `status: 'pending' | 'active' | 'banned' | 'closed'` - 用户状态。
        - `role: 'owner' | 'admin' | 'editor' | 'normal'` - 用户角色。
        - `last_login_at?: string` - 最后登录时间（可选）。
        - `last_active_at?: string` - 最后活跃时间（可选）。
        - `created_at?: string` - 创建时间（可选）。
- **后端交互:**
  - **URL:** {url}（`/workspaces/current/members: web/app/components/header/account-setting/members-page/index.tsx line 37`）
  - **方法:** GET

13. fetchProviders

- **用途:**用于获取可用的服务提供商列表。通过发送 GET 请求，从服务器获取当前可用的服务提供商信息。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含查询服务提供商信息的必要参数。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取服务提供商列表，解析后提供响应数据。
  - **数据类型:**
    - `Provider[] | null` - 响应数据类型。
      - `Provider` 详细信息见 `dify/web/models/common.ts line 94`。
- **后端交互:**
  - **URL:** {url}
  - **方法:** GET

14. validateProviderKey

- **用途:**用于验证服务提供商的 API Key。通过发送 POST 请求，向服务器提交 API Key 数据，并获取验证结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: { token: string }` - 请求体，包含需要验证的 API Key。
- **返回值:**
  - **描述:** 返回一个 Promise，用于验证 API Key，解析后提供响应数据。
  - **数据类型:**
    - `ValidateOpenAIKeyResponse` - 响应数据类型。
      - `result: string` - 验证结果。
      - `error?: string` - 错误信息（可选）。
- **后端交互:**
  - **URL:** {url}
  - **方法:** POST

15. updateProviderAIKey

- **用途:** 用于更新服务提供商的 API Key。通过发送 POST 请求，向服务器提交新的 API Key 数据，并获取更新结果。

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: { token: string | ProviderAzureToken | ProviderAnthropicToken }` - 请求体，包含新的 API Key 数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新 API Key，解析后提供响应数据。
  - **数据类型:**
    - `UpdateOpenAIKeyResponse` - 响应数据类型。
      - `result: string` - 更新结果。
      - `error?: string` - 错误信息（可选）。
- **后端交互:**
  - **URL:** {url}
  - **方法:** POST

16. fetchAccountIntegrates

- **用途:** 用于获取账户集成信息。通过发送 GET 请求，从服务器获取当前支持的账户登陆方式。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含查询账户集成信息所需要的参数。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取账户集成信息，解析后提供响应数据。
  - **数据类型:**
    - `{ data: AccountIntegrate[] | null }` - 响应数据类型。
      - `AccountIntegrate[]` - 账户集成信息列表。
        - `provider: 'google' | 'github'` - 集成服务提供商。
        - `created_at: number` - 创建时间。
        - `is_bound: boolean` - 是否绑定。
        - `link: string` - 绑定链接。
- **后端交互:**
  - **URL:** {url}（`/account/integrates: web/app/components/header/account-setting/Integrations-page/index.tsx line 28`）
  - **方法:** GET

17. inviteMember

- **用途:** 用于邀请新成员加入团队。通过发送 POST 请求，向服务器提交邀请数据，并获取邀请结果。

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含邀请数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于邀请新成员，解析后提供响应数据。
  - **数据类型:**
    - `InvitationResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
      - `invitation_results: InvitationResult[]` - 邀请结果列表。
        - `status: 'success'` – 邀请成功状态。
        - `email: string` - 邀请的邮箱地址。
        - `url: string` – 用于接受邀请的链接。

        or:

        - `status: 'failed'` – 邀请失败状态
        - `email: string` - 邀请的邮箱地址。
        - `message: string` – 邀请失败信息。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/members/invite-email: web/app/components/header/account-setting/members-page/invite-modal/index.tsx line 54`)
  - **方法:** POST

18. updateMemberRole

- **用途:** 用于更新团队成员的角色。通过发送 PUT 请求，向服务器提交更新数据，并获取更新结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含要更新的角色数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新成员角色，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/members/${member.id}/update-role: web/app/components/header/account-setting/members-page/operation/index.tsx line 55`)
  - **方法:** PUT

19. deleteMemberOrCancelInvitation

- **用途:** 用于删除团队成员或取消邀请。通过发送 DELETE 请求，向服务器提交删除或取消的数据，并获取处理结果。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于删除成员或取消邀请，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - URL: {url}(`/workspaces/current/members/${member.id}`: `web/app/components/header/account-setting/members-page/operation/index.tsx line 45`
  - **方法:** DELETE

20. fetchFilePreview

- **用途:** 用于获取文件预览内容。通过发送 GET 请求，从服务器获取指定文件的预览内容。
- **属性:**
  - `fileID: string` - 文件的 ID。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取文件预览内容，解析后提供响应数据。
  - **数据类型:**
    - `{ content: string }` - 响应数据类型。
      - `content: string` - 文件预览内容。
- **后端交互:**
  - **URL:** `/files/{fileID}/preview`
  - **方法:** GET

21. fetchCurrentWorkspace

- **用途:** 用于获取当前用户所在工作区的信息。通过发送 GET 请求，从服务器获取当前工作区的详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含查询当前工作区信息的必要参数。

- **返回值:**
  - **描述:** 返回一个 Promise，用于获取当前工作区信息，解析后提供响应数据。
  - **数据类型:**
    - `ICurrentWorkspace` - 响应数据类型。
      - `id: string` - 工作区 ID。
      - `name: string` - 工作区名称。
      - `plan: string` - 工作区计划。
      - `status: string` - 工作区状态。
      - `created_at: number` - 创建时间。
      - `role: 'owner' | 'admin' | 'editor' | 'normal'` - 用户在工作区中的角色。
      - `providers: Provider[]` - 服务提供商列表。
        - `Provider` 详细信息见 `web/models/common.ts line 94`。
      - `in_trail: boolean` - 是否在试用期。
      - `trial_end_reason?: string` - 试用期结束原因（可选）。
      - `custom_config?: { remove_webapp_brand?: boolean; replace_webapp_logo?: string }` - 自定义配置（可选）。
- **后端交互:**
  - URL: {url} (`/workspaces/current: web/context/app-context.tsx line 84`)
  - **方法:** GET

22. updateCurrentWorkspace

- **用途:** 用于更新当前工作区的信息。通过发送 POST 请求，将更新后的工作区信息提交到服务器。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含要更新的工作区信息。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新当前工作区信息，解析后提供响应数据。
  - **数据类型:**
    - `ICurrentWorkspace` - 响应数据类型(见上一项 21)。
- **后端交互:**
  - **URL:** {url} (`/workspaces/custom-config: web/app/components/custom/custom-web-app-brand/index.tsx line71/84/95`)
  - **方法:** POST

23. fetchWorkspaces

- **用途:** 用于获取用户所有工作区的信息。通过发送 GET 请求，从服务器获取用户所有工作区的详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: Record<string, any>` - 请求参数，包含查询工作区信息需要的参数。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取用户所有工作区的信息，解析后提供响应数据。
  - **数据类型:**
    - `{ workspaces: IWorkspace[] }` - 响应数据类型。
      - `workspaces: IWorkspace[]` - 工作区列表。
        - `id: string` - 工作区 ID。
        - `name: string` - 工作区名称。
        - `plan: string` - 工作区计划。
        - `status: string` - 工作区状态。
        - `created_at: number` - 创建时间。
        - `current: boolean` - 是否为当前工作区。
- **后端交互:**
  - **URL:** {url} (`/workspaces/: web/context/workspace-context.tsx line23`)
  - **方法:** GET

24. switchWorkspace

- **用途:** 用于切换当前工作区。通过发送 POST 请求，将用户切换到指定的工作区，并获取切换后的工作区信息。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: Record<string, any>` - 请求体，包含要切换的工作区信息。
- **返回值:**
  - **描述:** 返回一个 Promise，用于切换当前工作区，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse & { new_tenant: IWorkspace }` - 响应数据类型。
      - `CommonResponse`
        - `result: 'success' | 'fail'` - 请求的结果。
      - `new_tenant: IWorkspace` - 新的工作区信息。
        - `id: string` - 工作区 ID。
        - `name: string` - 工作区名称。
        - `plan: string` - 工作区计划。
        - `status: string` - 工作区状态。

        - `created_at: number` - 创建时间。
        - `current: boolean` - 是否为当前工作区。
- **后端交互:**
    - **URL:** {url} (`/workspaces/switch: web/app/components/header/account-dropdown/workplace-selector/index.tsx line36`)
    - **方法:** POST

25. fetchDataSource

- **用途:** 用于获取数据来源的信息。通过发送 GET 请求，从服务器获取指定数据来源的详细信息。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取数据来源信息，解析后提供响应数据。
    - **数据类型:**
        - `{ data: DataSourceNotion[] }` - 响应数据类型。
            - `data: DataSourceNotion[]` - 数据来源列表。
                - `id: string` - 数据来源 ID。
                - `provider: string` - 数据来源提供商。
                - `is_bound: boolean` - 是否绑定。
                - `source_info: DataSourceNotionWorkspace` - 数据来源详细信息。
                    - `workspace_name: string` - 工作区名称。
                    - `workspace_id: string` - 工作区 ID。
                    - `workspace_icon: string | null` - 工作区图标。
                    - `total?: number – 工作区总页数?`
                    - `pages: DataSourceNotionPage[]` - 页面列表。
                        - `page_id: string` - 页面 ID。
                        - `page_name: string` - 页面名称。
                        - `parent_id: string` - 父级 ID。
                        - `type: string` - 页面类型。
                        - `is_bound: boolean` - 是否绑定。
                        - `page_icon: null | {`

                          `type: string | null`

                          `url: string | null`

                          `emoji: string | null`

                          `}` – 页面图标

- **后端交互:**
  - **URL:** {url} (`data-source/integrates: web/app/components/header/account-setting/data-source-page/index.tsx line9`)
  - **方法:** GET

26. syncDataSourceNotion

- **用途:** 用于同步 Notion 数据源。通过发送 GET 请求，从服务器同步指定 Notion 数据源的信息。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于同步 Notion 数据源，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/oauth/data-source/notion/${payload.id}/sync: web/app/components/header/account-setting/data-source-page/data-source-notion/operate/index.tsx line44`)
  - **方法:** GET

27. updateDataSourceNotionAction

- **用途:** 用于更新 Notion 数据源的操作。通过发送 PATCH 请求，向服务器提交更新操作，并获取更新结果。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新 Notion 数据源操作，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/data-source/integrates/${payload.id}/disable: web/app/components/header/account-setting/data-source-page/data-source-notion/operate/index.tsx line48`)
  - **方法:** PATCH

28. fetchPluginProviders

- **用途:** 用于获取插件提供商的信息。通过发送 GET 请求，从服务器获取插件提供商的详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取插件提供商信息，解析后提供详细的插件提供商信息。
  - **数据类型:**
    - `PluginProvider[] | null` - 响应数据类型。
      - `PluginProvider[]` - 插件提供商详细信息列表
        - `tool_name: string` - 插件名
        - `is_enabled: Boolean` - 是否启用
        - `credentials: {` - 资格认证
          `api_key: string`
          `} | null`
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/tool-providers: web/app/components/header/account-setting/plugin-page/index.tsx line11`)
  - **方法:** GET

29. validatePluginProviderKey

- **用途:** 用于验证插件提供商的 API Key。通过发送 POST 请求，向服务器提交 API Key 数据，并获取验证结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: { credentials: any }` - 请求体，包含需要验证的 API Key。
- **返回值:**
  - **描述:** 返回一个 Promise，用于验证 API Key，解析后提供验证结果。
  - **数据类型:**
    - `ValidateOpenAIKeyResponse` - 响应数据类型。
      - `result: string` - 验证结果。
      - `error?: string` - 错误信息（可选）。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/tool-providers/${pluginType}/credentials-validate: web/app/components/header/account-setting/plugin-page/utils.tsx line7`)
  - **方法:** POST

30. updatePluginProviderAIKey

- **用途:** 用于更新插件提供商的 API Key。通过发送 POST 请求，向服务器提交新的 API Key 数据，并获取更新结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: { credentials: any }` - 请求体，包含新的 API Key 数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新 API Key，解析后提供响应数据。
  - **数据类型:**
    - `UpdateOpenAIKeyResponse` - 响应数据类型。
      - `result: string` - 更新结果。
      - `error?: string` - 错误信息（可选）。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/tool-providers/${pluginType}/credentials: web/app/components/header/account-setting/plugin-page/utils.tsx line23`)
  - **方法:** POST

31. invitationCheck

- **用途:** 用于验证激活邀请的有效性。它确保提供的邀请参数（工作区 ID、电子邮件和令牌）正确无误，并且邀请仍处于活动状态。通过发送 GET 请求，向服务器提交邀请链接的相关信息，并获取验证结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `params: { workspace_id: string; email: string; token: string }` - 请求参数，包含邀请链接的必要数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于检查邀请链接的有效性，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse & { is_valid: boolean; workspace_name: string }` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
      - `is_valid: boolean` - 邀请链接是否有效。
      - `workspace_name: string` - 工作区名称。
- **后端交互:**
  - **URL:** {url} (`/activate/check: web/app/activate/activateForm.tsx line38 -> 31`)
  - **方法:** GET

32. activateMember

- **用途:** 用于激活成员账户。通过发送 POST 请求，向服务器提交激活数据，并获取激活结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: any` - 请求体，包含激活数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于激活成员账户，解析后提供响应数据。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/activate: web/app/activate/activateForm.tsx line77`)
  - **方法:** POST

33. fetchModelProviders

- **用途:** 用于获取模型提供商的信息。通过发送 GET 请求，从服务器获取模型提供商的详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取模型提供商信息，解析后提供响应数据。
  - **数据类型:**
    - `{ data: ModelProvider[] }` - 响应数据类型。
      - `data: ModelProvider[]` - 模型提供商列表。
        - `ModelProvider` 详细信息见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line 151`。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers: web/app/components/app/configuration/toolbox/moderation/moderation-setting-modal.tsx line47`)
  - **方法:** GET

34. fetchModelProviderCredentials

- **用途:** 用于获取模型提供商的凭据信息。通过发送 GET 请求，从服务器获取模型提供商的凭据详细信息。

- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取模型提供商的凭据信息，解析后提供响应数据。
  - **数据类型:**
    - `ModelProviderCredentials` - 响应数据类型。
      - `credentials?: Record<string, string | undefined | boolean>` - 凭据信息。
      - `load_balancing: ModelLoadBalancingConfig` – 模型负载均衡配置。
        - `ModelLoadBalancingConfig` 详细信息见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line 244`。
- **后端交互:**
  - **URL:** {url} (`web/app/components/header/account-setting/model-provider-page/hooks.ts line72/78`)
  - **方法:** GET

35. fetchModelLoadBalancingConfig

- **用途:** 用于获取模型负载均衡配置。通过发送 GET 请求，从服务器获取模型的负载均衡详细配置。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取模型负载均衡配置，解析后提供响应数据。
  - **数据类型:**
    - `Object` - 响应数据类型。
      - `credentials?: Record<string, string | undefined | boolean>` - 凭据信息。
      - `load_balancing: ModelLoadBalancingConfig` - 负载均衡配置。
        - `ModelLoadBalancingConfig` 详细信息见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line 244`。
- **后端交互:**
  - **URL:** {url} (`web/app/components/header/account-setting/model-provider-page/provider-added-card/model-load-balancing-modal.tsx line33`)。
  - **方法:** GET

36. fetchModelProviderModelList

- **用途:** 用于获取模型提供商的模型列表。通过发送 GET 请求，从服务器获取指定模型提供商的模型详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取模型列表，解析后提供响应数据。
  - **数据类型:**
    - `{ data: ModelItem[] }` - 响应数据类型。
      - `data: ModelItem[]` - 模型列表。
        - `ModelItem` 详细信息见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line113`。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers/${providerName}/models: web/app/components/header/account-setting/model-provider-page/provider-added-card/index.tsx line54`)
  - **方法:** GET

37. fetchModelList

- **用途:** 用于获取所有模型的列表。通过发送 GET 请求，从服务器获取所有模型的详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取模型列表，解析后提供响应数据。
  - **数据类型:**
    - `{ data: Model[] }` - 响应数据类型。
      - `data: Model[]` - 模型列表。
        - `Model` 详细信息见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line185`。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/models/model-types/${type}: web/app/components/header/account-setting/model-provider-page/hooks.ts line116`)
  - **方法:** GET

38. validateModelProvider

- **用途:** 用于验证模型提供商的 API Key。通过发送 POST 请求，向服务器提交 API Key 数据，并获取验证结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: any` - 请求体，包含需要验证的 API Key。
- **返回值:**
  - **描述:** 返回一个 Promise，用于验证 API Key，解析后提供验证结果。
  - **数据类型:**
    - `ValidateOpenAIKeyResponse` - 响应数据类型。
      - `result: string` - 验证结果。
      - `error?: string` - 错误信息（可选）。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers/${provider}/models/credentials/validate: web/app/components/header/account-setting/model-provider-page/utils.ts line45`)
  - **方法:** POST

39. validateModelLoadBalancingCredentials

- **用途:** 用于验证模型负载均衡的凭据信息。通过发送 POST 请求，向服务器提交负载均衡凭据数据，并获取验证结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: any` - 请求体，包含需要验证的负载均衡凭据数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于验证负载均衡凭据，解析后提供验证结果。
  - **数据类型:**
    - `ValidateOpenAIKeyResponse` - 响应数据类型。
      - `result: string` - 验证结果。
      - `error?: string` - 错误信息（可选）。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers/${provider}/models/load-balancing-configs/credentials-validate: web/app/components/header/account-setting/model-provider-page/utils.ts line66`)
  - **方法:** POST

40. setModelProvider

- **用途:** 用于设置模型提供商的配置。通过发送 POST 请求，向服务器提交新的模型提供商配置，并获取设置结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: any` - 请求体，包含新的模型提供商配置。
- **返回值:**
  - **描述:** 返回一个 Promise，用于设置模型提供商配置，解析后提供设置结果。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers/${provider}(/models): web/app/components/header/account-setting/model-provider-page/utils.ts line92/102`)
  - **方法:** POST

41. deleteModelProvider

- **用途:** 用于删除模型提供商。通过发送 DELETE 请求，向服务器提交删除数据，并获取删除结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body?: any` - 可选的请求体，包含删除数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于删除模型提供商，解析后提供删除结果。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers/${provider}(/models): web/app/components/header/account-setting/model-provider-page/utils.ts line127/136`)
  - **方法:** DELETE

42. changeModelProviderPriority

- **用途:** 用于更改模型提供商的优先级。通过发送 POST 请求，向服务器提交新的优先级数据，并获取更改结果。

- **属性:**
  - `url: string` - 请求的 URL。
  - `body: any` - 请求体，包含新的优先级数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更改模型提供商的优先级，解析后提供更改结果。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url} (`/workspaces/current/model-providers/${provider.provider}/preferred-provider-type: web/app/components/header/account-setting/model-provider-page/provider-added-card/credential-panel.tsx line44`)
  - **方法:** POST

43. setModelProviderModel

- **用途:** 用于设置模型提供商的模型配置。通过发送 POST 请求，向服务器提交新的模型配置，并获取设置结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: any` - 请求体，包含新的模型配置。
- **返回值:**
  - **描述:** 返回一个 Promise，用于设置模型配置，解析后提供设置结果。
  - **数据类型:**
    - `CommonResponse` - 响应数据类型。
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
  - **URL:** {url}
  - **方法:** POST

44. deleteModelProviderModel

- **用途:** 用于删除模型提供商的模型配置。通过发送 DELETE 请求，向服务器提交删除数据，并获取删除结果。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于删除模型配置，解析后提供删除结果。

  - **数据类型:**

        - `CommonResponse` - 响应数据类型。
            - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
    - **URL:** {url}
    - **方法:** DELETE

45. getPayUrl

- **用途:** 用于获取支付 URL。通过发送 GET 请求，从服务器获取支付 URL。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取支付 URL，解析后提供支付 URL。
    - **数据类型:**
        - `{ url: string }` - 响应数据类型。
            - `url: string` - 支付 URL。
- **后端交互:**
    - **URL:** {url} (`/workspaces/current/model-providers/anthropic/checkout-url: web/app/components/header/account-setting/model-provider-page/hooks.ts line 205`)
    - **方法:** GET

46. fetchDefaultModal

- **用途:** 用于获取默认模型的配置信息。通过发送 GET 请求，从服务器获取默认模型的详细信息。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取默认模型的配置信息，解析后提供响应数据。
    - **数据类型:**
        - `{ data: DefaultModelResponse }` - 响应数据类型。
            - `data: DefaultModelResponse` - 默认模型信息，详细见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line194`
                - `model: string` - 模型名称。
                - `model_type: ModelTypeEnum` - 模型类型。
                - `provider: { provider: string; icon_large: TypeWithI18N; icon_small: TypeWithI18N }` - 提供商信息。

- **后端交互:**

    - **URL:** {url} (`/workspaces/current/default-model?model_type=${type}: web/app/components/header/account-setting/model-provider-page/hooks.ts line 126`)
    - **方法:** GET

47. updateDefaultModel

- **用途:** 用于更新默认模型的配置信息。通过发送 POST 请求，向服务器提交新的默认模型配置，并获取更新结果。
- **属性:**
    - `url: string` - 请求的 URL。
    - `body: any` - 请求体，包含新的默认模型配置。
- **返回值:**
    - **描述:** 返回一个 Promise，用于更新默认模型的配置信息，解析后提供更新结果。
    - **数据类型:**
        - `CommonResponse` - 响应数据类型。
            - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
    - **URL:** {url} (`/workspaces/current/default-model: web/app/components/header/account-setting/model-provider-page/system-model-selector/index.tsx line 93`)
    - **方法:** POST

48. fetchModelParameterRules

- **用途:** 用于获取模型参数的规则。通过发送 GET 请求，从服务器获取模型参数的详细规则。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取模型参数的规则，解析后提供响应数据。
    - **数据类型:**
        - `{ data: ModelParameterRule[] }` - 响应数据类型。
            - `data: ModelParameterRule[]` - 模型参数规则列表。
                - `ModelParameterRule` 详细信息见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line 214`。

- **后端交互:**

    - **URL:** {url} (`/workspaces/current/model-providers/${provider}/models/parameter-rules?model=${modelId}: web/app/components/header/account-setting/model-provider-page/model-parameter-modal/index.tsx line 91`)
    - **方法:** GET

49. fetchFileUploadConfig

- **用途:** 用于从服务器获取与文件上传相关的配置信息，包括文件大小限制和批量上传数量限制。此函数有助于前端工程师在实现文件上传功能时确保文件上传符合服务器的要求，并在上传文件前进行必要的检查，以避免因文件大小或数量超出限制而导致的上传失败。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取文件上传的配置信息，解析后提供详细的上传配置。
    - **数据类型:**
        - `FileUploadConfigResponse` - 响应数据类型。
            - `file_size_limit: number` - 文件大小限制。
            - `batch_count_limit: number` - 批量上传数量限制。
            - `image_file_size_limit?: number | string` - 图片文件大小限制（可选）。
- **后端交互:**
    - **URL:** {url} (`/files/upload: web/app/components/datasets/create/file-uploader/index.tsx line 48`)
    - **方法:** GET

50. fetchFreeQuotaVerify

- **用途:** 用于从服务器验证用户的免费使用额度。这包括剩余的免费使用次数或令牌等信息。前端工程师可以使用此函数在应用中实现使用额度限制功能，例如在用户尝试进行某些操作前，先检查其是否有足够的免费额度，从而确保应用内的资源使用在可控范围内。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取免费额度验证信息，解析后提供详细的验证结果。
    - **数据类型:**
        - `Object` - 响应数据类型。
            - `result: string` - 验证结果。

        - `flag: boolean` - 验证标志。
        - `reason: string` - 验证原因。
- **后端交互:**
    - **URL:** {url} (`/workspaces/current/model-providers/${provider}/free-quota-qualification-verify?token=${token}: web/hooks/use-pay.tsx line 92`)
    - **方法:** GET

51. fetchNotionConnection

- **用途:** 用于从服务器获取 Notion 数据源的连接信息。可以获取当前 Notion 数据源的连接状态和详细信息，从而在用户界面上准确显示 Notion 数据源的使用情况和连接状态。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取 Notion 数据源连接信息，解析后提供详细的连接状态和信息。
    - **数据类型:**
        - `{ data: string }` - 响应数据类型。
            - `data: string` - Notion 数据源连接信息。
- **后端交互:**
    - **URL:** {url} (`/oauth/data-source/notion: web/app/components/header/account-setting/data-source-page/data-source-notion/index.tsx line 34`)
    - **方法:** GET

52. fetchDataSourceNotionBinding

- **用途:** 用于从服务器获取 Notion 数据源的绑定信息。使用此函数在应用中查看和管理 Notion 数据源的绑定状态，确保用户能够清晰地了解哪些 Notion 数据源已经绑定以及其绑定的详细信息。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取 Notion 数据源绑定信息，解析后提供详细的绑定状态和信息。
    - **数据类型:**
        - `{ result: string }` - 响应数据类型。
            - `result: string` - Notion 数据源绑定信息。

- **后端交互:**

    - **URL:** {url} (`/oauth/data-source/binding/notion?code=${notionCode}: web/hooks/use-pay.tsx line 126`)
    - **方法:** GET

53. fetchApiBasedExtensionList

- **用途:** 用于从服务器获取所有基于 API 的扩展列表。使用此函数在应用中显示和管理这些扩展，帮助用户查看可用的 API 扩展，并选择适合其需求的扩展进行使用或配置。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取 API 基础扩展列表，解析后提供详细的扩展信息。
    - **数据类型:**
        - `ApiBasedExtension[]` - 响应数据类型。
            - `id?: string` - 扩展 ID（可选）。
            - `name?: string` - 扩展名称（可选）。
            - `api_endpoint?: string` - API 端点（可选）。
            - `api_key?: string` - API 密钥（可选）。
- **后端交互:**
    - **URL:** {url} (`/api-based-extension: web/app/components/header/account-setting/api-based-extension-page/index.tsx line 15`)
    - **方法:** GET

54. fetchApiBasedExtensionDetail

- **用途:** 用于从服务器获取特定 API 扩展的详细信息。使用此函数查看某个特定 API 扩展的详细信息。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取特定 API 扩展的详细信息，解析后提供详细的扩展信息。
    - **数据类型:**
        - `ApiBasedExtension` - 响应数据类型。
            - `id?: string` - 扩展 ID（可选）。
            - `name?: string` - 扩展名称（可选）。
            - `api_endpoint?: string` - API 端点（可选）。
            - `api_key?: string` - API 密钥（可选）。
- **后端交互:**
    - **URL:** {url}

- **方法:** GET

55. addApiBasedExtension

- **用途:** 用于添加新的 API 扩展。使用此函数通过发送 POST 请求，向服务器提交新扩展的数据，并获取添加结果。对于扩展应用的功能和集成第三方服务非常有用。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: ApiBasedExtension` - 请求体，包含新扩展的数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于添加新的 API 扩展，解析后提供添加结果。
  - **数据类型:**
    - `ApiBasedExtension` - 响应数据类型。
      - `id?: string` - 扩展 ID（可选）。
      - `name?: string` - 扩展名称（可选）。
      - `api_endpoint?: string` - API 端点（可选）。
      - `api_key?: string` - API 密钥（可选）。
- **后端交互:**
  - **URL:** {url} (`/api-based-extension: web/app/components/header/account-setting/api-based-extension-page/modal.tsx line 50`)
  - **方法:** POST

56. updateApiBasedExtension

- **用途:** 用于更新 API 扩展信息。使用此函数通过发送 POST 请求，向服务器提交更新后的扩展数据，并获取更新结果。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: ApiBasedExtension` - 请求体，包含更新后的扩展数据。
- **返回值:**
  - **描述:** 返回一个 Promise，用于更新 API 扩展信息，解析后提供更新结果。
  - **数据类型:**
    - `ApiBasedExtension` - 响应数据类型。
      - `id?: string` - 扩展 ID（可选）。
      - `name?: string` - 扩展名称（可选）。
      - `api_endpoint?: string` - API 端点（可选）。
      - `api_key?: string` - API 密钥（可选）。

- **后端交互:**

    - **URL:** {url} (`/api-based-extension/${data.id}: web/app/components/header/account-setting/api-based-extension-page/modal.tsx line 56`)
    - **方法:** POST

57. deleteApiBasedExtension

- **用途:** 用于删除 API 扩展。使用此函数通过发送 DELETE 请求，向服务器提交删除请求，并获取删除结果。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于删除 API 扩展，解析后提供删除结果。
    - **数据类型:**
        - `{ result: string }` - 响应数据类型。
            - `result: string` - 删除结果。
- **后端交互:**
    - **URL:** {url} (`/api-based-extension/${data.id}: web/app/components/header/account-setting/api-based-extension-page/item.tsx line 32`)
    - **方法:** DELETE

58. fetchCodeBasedExtensionList

- **用途:** 用于获取基于代码的扩展工具列表。使用此函数从服务器获取所有基于代码的扩展工具的详细信息。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取代码基础扩展列表，解析后提供详细的扩展信息。
    - **数据类型:**
        - `CodeBasedExtension` - 响应数据类型。
            - `module: string` - 模块名称。
            - `data: CodeBasedExtensionItem[]` - 扩展项列表。
                - `CodeBasedExtensionItem` 详细信息见 `web/models/common.ts line 250`。
- **后端交互:**
    - **URL:** {url} (`/code-based-extension(?arg:module=) Ex. /code-based-extension?module=moderation: web/app/components/app/configuration/toolbox/moderation/index.tsx line 19`)
    - **方法:** GET

59. moderate

- **用途:** 用于对给定文本进行内容审核。使用此函数通过发送 POST 请求，向服务器提交需要审核的文本，并获取审核结果。有利于确保应用内容的合规性和过滤不良信息。
- **属性:**
  - `url: string` - 请求的 URL。
  - `body: { app_id: string; text: string }` - 请求体，包含需要审核的文本和应用 ID。
- **返回值:**
  - **描述:** 返回一个 Promise，用于审核文本内容，解析后提供审核结果。
  - **数据类型:**
    - `ModerateResponse` - 响应数据类型。
      - `flagged: boolean` - 是否被标记为不良内容。
      - `text: string` - 审核后的文本。
- **后端交互:**
  - **URL:** {url}
  - **方法:** POST

60. fetchSupportRetrievalMethods

- **用途:** 用于获取支持的检索方法。使用此函数从服务器获取所有支持的检索方法的详细信息。
- **属性:**
  - `url: string` - 请求的 URL。
- **返回值:**
  - **描述:** 返回一个 Promise，用于获取支持的检索方法，解析后提供详细的检索方法信息。
  - **数据类型:**
    - `RetrievalMethodsRes` - 响应数据类型。
      - `retrieval_method: RETRIEVE_METHOD[]` - 检索方法列表。
        - `RETRIEVE_METHOD` – 检索方法。
          - `semantic: 'semantic_search'` - 语义搜索。
          - `fullText: 'full_text_search'` - 全文搜索。
          - `hybrid: 'hybrid_search'` - 混合搜索。
          - `invertedIndex: 'invertedIndex'` - 倒排索引。
          - `keywordSearch: 'keyword_search'` - 关键词搜索。
- **后端交互:**

    - **URL:** {url} (`/datasets/retrieval-setting: web/context/provider-context.tsx line 82`)
    - **方法:** GET

61. getSystemFeatures

- **用途:** 用于获取系统功能特性。使用此函数从服务器获取当前系统的所有功能特性的信息。
- **属性:**
    - `url: string` - 请求的 URL。
- **返回值:**
    - **描述:** 返回一个 Promise，用于获取系统功能特性，解析后提供详细的功能特性信息。
    - **数据类型:**
        - `SystemFeatures` - 响应数据类型。
            - `sso_enforced_for_signin: boolean` - 是否强制使用单点登录（SSO）进行登录。
            - `sso_enforced_for_signin_protocol: string` - 强制使用 SSO 登录的协议。
            - `sso_enforced_for_web: boolean` - 是否强制在 Web 上使用 SSO。
            - `sso_enforced_for_web_protocol: string` - 强制在 Web 上使用 SSO 的协议。
- **后端交互:**
    - **URL:** `/system-features: （web/app/signin/page.tsx line 21）`
    - **方法:** GET

62. enableModel

- **用途:** 用于启用指定的模型。可以使用此函数通过发送 PATCH 请求，向服务器提交需要启用的模型信息，并获取启用结果。
- **属性:**
    - `url: string` - 请求的 URL。
    - `body: { model: string; model_type: ModelTypeEnum }` - 请求体，包含需要启用的模型信息。
- **返回值:**
    - **描述:** 返回一个 Promise，用于启用模型，解析后提供启用结果。
    - **数据类型:**
        - `CommonResponse` - 响应数据类型。
            - `result: 'success' | 'fail'` - 请求的结果。

- **后端交互:**

    - **URL:** {url} (`/workspaces/current/model-providers/${provider.provider}/models/enable: web/app/components/header/account-setting/model-provider-page/provider-added-card/model-list-item.tsx line 34`)
    - **方法:** PATCH

63. disableModel

- **用途:** 用于禁用指定的模型。使用此函数通过发送 PATCH 请求，向服务器提交需要禁用的模型信息，并获取禁用结果。
- **属性:**
    - `url: string` - 请求的 URL。
    - `body: { model: string; model_type: ModelTypeEnum }` - 请求体，包含需要禁用的模型信息。
- **返回值:**
    - **描述:** 返回一个 Promise，用于禁用模型，解析后提供禁用结果。
    - **数据类型:**
        - `CommonResponse` - 响应数据类型。
            - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互:**
    - **URL:** {url} (`/workspaces/current/model-providers/${provider.provider}/models/disable: web/app/components/header/account-setting/model-provider-page/provider-added-card/model-list-item.tsx line 36`)
    - **方法:** PATCH

Web/service/datasets.ts – *用于“知识库”(Knowledge)管理，包括文档操作、索引、段处理等API接口*

1. fetchDatasetDetail

- **用途**: 用于获取特定数据集的详细信息。在前端显示和管理数据集的详细信息，包括数据集的基本属性和配置。
- **属性**:
  - `datasetId: string` - 数据集的唯一 ID。
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取数据集的详细信息，解析后提供数据集的完整属性和配置信息。
  - **数据类型**:
    - `DataSet` - 响应数据类型，包含以下属性：
      - `id: string` - 数据集的唯一标识符
      - `name: string` - 数据集的名称
      - `icon: string` - 数据集的图标
      - `icon_background: string` - 图标的背景颜色
      - `description: string` - 数据集的描述
      - `permission: 'only_me' | 'all_team_members'` - 数据集的权限设置
      - `data_source_type: 'upload_file' | 'notion_import' | 'website_crawl'` - 数据源类型
      - `indexing_technique: 'high_quality' | 'economy'` - 索引技术
      - `created_by: string` - 创建者
      - `updated_by: string` - 更新者
      - `updated_at: number` - 更新时间
      - `app_count: number` - 应用程序数量
      - `document_count: number` - 文档数量
      - `word_count: number` - 单词数量
      - `embedding_model: string` - 嵌入模型
      - `embedding_model_provider: string` - 嵌入模型提供者
      - `embedding_available: boolean` - 是否有嵌入
      - `retrieval_model_dict: RetrievalConfig` - 检索模型配置，详细见 `web/types/app.ts line 376`
      - `retrieval_model: RetrievalConfig` - 检索模型，详细见 `web/types/app.ts line 376`
      - `tags: Tag[]` - 标签列表
        - `id: string` - 标签的唯一标识符
        - `name: string` - 标签的名称

        - `type: string` - 标签的类型
        - `binding_count: number` - 绑定数量
- **后端交互**:
    - **URL**: `/datasets/${datasetId}`
    - **方法**: `GET`

2. updateDatasetSetting

- **用途**: 用于更新特定数据集的设置信息。在前端修改和保存数据集的名称、描述、权限和模型设置等。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `body: Partial<Pick<DataSet, 'name' | 'description' | 'permission' | 'indexing_technique' | 'retrieval_model' | 'embedding_model' | 'embedding_model_provider'>>` - 需要更新的字段及其新值
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取更新后的数据集信息，解析后提供数据集的更新属性和配置信息。
    - **数据类型**:
        - `DataSet` - 同 `1. fetchDatasetDetail` 中的 `DataSet` 类型
- **后端交互**:
    - **URL**: `/datasets/${datasetId}`
    - **方法**: `PATCH`

3. fetchDatasetRelatedApps

- **用途**: 用于获取与特定数据集相关的应用信息。在前端显示与该数据集相关的所有应用。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取与数据集相关的应用信息，解析后提供相关应用的详细信息。
    - **数据类型**:
        - `RelatedAppResponse` - 响应数据类型，包括以下属性：
            - `data: Array<RelatedApp>` - 相关应用列表，包括以下属性：
                - `id: string` - 应用的唯一标识符
                - `name: string` - 应用名称
                - `mode: AppMode` - 应用模式(可能值为 `advanced-chat, agent-chat, chat, completion, workflow`)
                - `icon: string` - 应用图标
                - `icon_background: string` - 图标背景颜色

        - `total: number` - 总数
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/related-apps`
    - **方法**: `GET`

4. fetchDatasets

- **用途**: 用于获取数据集列表。在前端显示所有数据集的信息，支持分页和筛选。
- **属性**:
    - `url: string` - 请求的 URL
    - `params: { page: number; ids?: string[]; limit?: number }` - 请求参数，包括页码、数据集 ID 列表（可选）和每页数量（可选）
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取数据集列表，解析后提供数据集的详细信息。
    - **数据类型**:
        - `DataSetListResponse` - 响应数据类型，包括以下属性：
            - `data: DataSet[]` - 数据集列表，每个数据集为 `DataSet` 类型，同 `1. fetchDatasetDetail` 中的 `DataSet` 类型
            - `has_more: boolean` - 是否有更多数据
            - `limit: number` - 每页数量
            - `page: number` - 当前页码
            - `total: number` - 数据集总数
- **后端交互**:
    - **URL**: `/datasets`
    - **方法**: `GET`

5. createEmptyDataset

- **用途**: 用于创建一个空的数据集。在前端初始化一个新的数据集。
- **属性**:
    - `name: string` - 数据集的名称
- **返回值**:
    - **描述**: 返回一个 Promise，用于创建数据集，解析后提供新创建的数据集详细信息。
    - **数据类型**:
        - `DataSet` - 同 `1. fetchDatasetDetail` 中的 `DataSet` 类型
- **后端交互**:
    - **URL**: `/datasets`
    - **方法**: `POST`

6. deleteDataset

- **用途**: 用于删除特定数据集。在前端移除不再需要的数据集。
- **属性**:
  - `datasetID: string` - 数据集的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于删除数据集，解析后提供操作结果。
  - **数据类型**:
    - `DataSet` - 同 `1. fetchDatasetDetail` 中的 `DataSet` 类型
- **后端交互**:
  - **URL**: `/datasets/${datasetID}`
  - **方法**: `DELETE`

7. fetchDefaultProcessRule

- **用途**: 用于获取默认的处理规则。在前端显示和管理默认的文档处理规则。
- **属性**:
  - `url: string` - 请求的 URL
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取默认的处理规则，解析后提供规则的详细信息。
  - **数据类型**:
    - `ProcessRuleResponse` - 响应数据类型，包括以下属性（详细见 `dify/web/models/datasets.ts line 110`）：
      - `mode: ProcessMode` - 处理模式
        - `ProcessMode`
          - `'automatic' | 'custom'`
      - `rules: Rules` - 规则集，包含预处理规则和分割规则
- **后端交互**:
  - **URL**: `{url}`
    - 用例：`/datasets/process-rule: web/app/components/datasets/create/step-two/index.tsx line 381`
  - **方法**: `GET`

8. fetchProcessRule

- **用途**: 用于获取特定文档的处理规则。
- **属性**:
  - `params: { documentId: string }` - 请求参数，包括文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取处理规则，解析后提供规则的详细信息。

    - **数据类型**:
        - `ProcessRuleResponse` - 同 `7. fetchDefaultProcessRule` 中的 `ProcessRuleResponse` 类型
- **后端交互**:
    - **URL**: `/datasets/process-rule`
    - **方法**: `GET`

9. fetchDocuments

- **用途**: 用于获取特定数据集中的文档列表。在前端显示和管理数据集中的所有文档。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `params: { keyword: string; page: number; limit: number; sort?: SortType }` - 请求参数，包括关键词、页码、每页数量和排序类型（可选）
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取文档列表，解析后提供文档的详细信息。
    - **数据类型**:
        - `DocumentListResponse` - 响应数据类型，包括以下属性：
            - `data: SimpleDocumentDetail[]` - 文档列表，每个文档为 `SimpleDocumentDetail` 类型
                - `SimpleDocumentDetail` 详见 `dify/web/models/datasets.ts line 191`
            - `has_more: boolean` - 是否有更多数据
            - `total: number` - 文档总数
            - `page: number` - 当前页码
            - `limit: number` - 每页数量
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/documents`
    - **方法**: `GET`

10. createFirstDocument

- **用途**: 用于在数据集中创建第一个文档。在前端初始化数据集时创建第一个文档。
- **属性**:
    - `body: CreateDocumentReq` - 请求体，包括文档创建所需的所有信息
- **返回值**:
    - **描述**: 返回一个 Promise，用于创建文档，解析后提供新创建的文档详细信息。

    - **数据类型**:
        - `createDocumentResponse` - 响应数据类型，包括以下属性：
            - `dataset?: DataSet` - 相关数据集，类型同 `1. fetchDatasetDetail` 中的 `DataSet`
            - `batch: string` - 批次标识符
            - `documents: InitialDocumentDetail[]` - 文档列表，每个文档为 `InitialDocumentDetail` 类型
                - `InitialDocumentDetail` 详细见 `dify/web/models/datasets.ts line 172`
- **后端交互**:
    - **URL**: `/datasets/init`
    - **方法**: `POST`

11. createDocument

- **用途**: 用于在数据集中创建一个新文档。在前端向现有数据集添加文档。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `body: CreateDocumentReq` - 请求体，包括文档创建所需的所有信息
- **返回值**:
    - **描述**: 返回一个 Promise，用于创建文档，解析后提供新创建的文档详细信息。
    - **数据类型**:
        - `createDocumentResponse` - 同 `10. createFirstDocument` 中的 `createDocumentResponse` 类型
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/documents`
    - **方法**: `POST`

12. fetchIndexingEstimate

- **用途**: 用于获取特定文档的索引估算信息。在前端显示文档的索引估算，包括预估的处理时间和费用。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `documentId: string` - 文档的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取索引估算信息，解析后提供估算的详细信息。
    - **数据类型**:
        - `IndexingEstimateResponse` - 响应数据类型，包括以下属性：

        - `tokens: number` - 估算的令牌数
        - `total_price: number` - 总价格
        - `currency: string` - 货币类型
        - `total_segments: number` - 总段数
        - `preview: string[]` - 预览
        - `qa_preview?: QA[]` - 问答预览（可选）
            - `question: string` - 问题
            - `answer: string` - 答案
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/documents/${documentId}/indexing-estimate`
    - **方法**: `GET`

13. fetchIndexingEstimateBatch

- **用途**: 用于获取一批文档的索引估算信息。在前端显示一批文档的索引估算，包括预估的处理时间和费用。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `batchId: string` - 批次的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取索引估算信息，解析后提供估算的详细信息。
    - **数据类型**:
        - `IndexingEstimateResponse` - 响应数据类型
            - 同 `12. fetchIndexingEstimate` 中的 `IndexingEstimateResponse` 类型
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/batch/${batchId}/indexing-estimate`
    - **方法**: `GET`

14. fetchIndexingStatus

- **用途**: 用于获取特定文档的索引状态。在前端显示文档的当前索引状态。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `documentId: string` - 文档的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取文档的索引状态，解析后提供详细信息。
    - **数据类型**:
        - `IndexingStatusResponse` - 响应数据类型，包括以下属性：

        - `id: string` - 文档的唯一标识符
        - `indexing_status: DocumentIndexingStatus` - 索引状态
            - 可能值有 `waiting, parsing, cleaning, splitting, indexing, paused, error, completed`
        - `processing_started_at: number` - 处理开始时间
        - `parsing_completed_at: number` - 解析完成时间
        - `cleaning_completed_at: number` - 清理完成时间
        - `splitting_completed_at: number` - 分割完成时间
        - `completed_at: any` - 完成时间
        - `paused_at: any` - 暂停时间
        - `error: any` - 错误信息
        - `stopped_at: any` - 停止时间
        - `completed_segments: number` - 完成的段数
        - `total_segments: number` - 总段数
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/documents/${documentId}/indexing-status`
    - **方法**: `GET`

15. fetchIndexingStatusBatch

- **用途**: 用于获取一批文档的索引状态。在前端显示一批文档的当前索引状态。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `batchId: string` - 批次的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取一批文档的索引状态，解析后提供详细信息。
    - **数据类型**:
        - `IndexingStatusBatchResponse` - 响应数据类型，包括以下属性：
            - `data: IndexingStatusResponse[]` - 文档索引状态列表，每个文档的索引状态为 `IndexingStatusResponse` 类型
            - 同 `14. fetchIndexingStatus` 中的 `IndexingStatusResponse` 类型
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/batch/${batchId}/indexing-status`
    - **方法**: `GET`

16. fetchDocumentDetail

- **用途**: 用于获取特定文档的详细信息。在前端显示和管理文档的详细信息，包括文档的元数据和内容。

- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
  - `params: { metadata?: MetadataType }` - 请求参数，包括元数据类型（可选）
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取文档的详细信息，解析后提供文档的完整属性和配置信息。
  - **数据类型**:
    - `DocumentDetailResponse` - 响应数据类型
      - 详细见 `web/models/datasets.ts line 320 -> 268`
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}`
  - **方法**: `GET`

17. renameDocumentName

- **用途**: 用于重命名特定文档的名称。在前端修改文档的名称。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
  - `name: string` - 新的文档名称
- **返回值**:
  - **描述**: 返回一个 Promise，用于重命名文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型
      - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/rename`
  - **方法**: `POST`

18. pauseDocIndexing

- **用途**: 用于暂停特定文档的索引。在前端暂停文档的索引处理。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于暂停文档索引，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果

- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/processing/pause`
  - **方法**: `PATCH`

19. resumeDocIndexing

- **用途**: 用于恢复特定文档的索引。在前端恢复文档的索引处理。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于恢复文档索引，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/processing/resume`
  - **方法**: `PATCH`

20. deleteDocument

- **用途**: 用于删除特定文档。在前端移除不再需要的文档。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于删除文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}`
  - **方法**: `DELETE`

21. archiveDocument

- **用途**: 用于将特定文档归档。在前端将文档状态设置为归档。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符

- **返回值**:
  - **描述**: 返回一个 Promise，用于归档文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/status/archive`
  - **方法**: `PATCH`

22. unArchiveDocument

- **用途**: 用于将特定文档从归档状态恢复。前端将文档状态从归档恢复为正常。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于恢复归档文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/status/un_archive`
  - **方法**: `PATCH`

23. enableDocument

- **用途**: 用于启用特定文档。在前端将文档状态设置为启用。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于启用文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/status/enable`
  - **方法**: `PATCH`

24. disableDocument

- **用途**: 用于禁用特定文档。在前端将文档状态设置为禁用。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于禁用文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/status/disable`
  - **方法**: `PATCH`

25. syncDocument

- **用途**: 用于同步特定文档。在前端将文档与外部数据源进行同步，`if data_source_type === 'notion_import'`。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于同步文档，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/notion/sync`
  - **方法**: `GET`

26. syncWebsite

- **用途**: 用于同步特定网站文档。在前端将网站文档与外部数据源进行同步。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于同步网站文档，解析后提供操作结果。

    - **数据类型**:
        - `CommonResponse` - 响应数据类型，包括操作结果信息
            - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/documents/${documentId}/website-sync`
    - **方法**: `GET`

27. preImportNotionPages

- **用途**: 用于预导入 Notion 页面。在前端获取 Notion 页面信息，以便选择和导入页面。
- **属性**:
    - `url: string` - 请求的 URL
    - `datasetId?: string` - 数据集的唯一标识符（可选）
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取 Notion 页面信息，解析后提供页面详细信息。
    - **数据类型**:
        - `{ notion_info: DataSourceNotionWorkspace[] }` - 响应数据类型，包括 Notion 工作区信息
            - `DataSourceNotionWorkspace` - 类型，包括以下属性：
                - `workspace_name: string` - 工作区名称
                - `workspace_id: string` - 工作区的唯一标识符
                - `workspace_icon: string | null` - 工作区图标
                - `total?: number` - 总数（可选）
                - `pages: DataSourceNotionPage[]` - 页面列表
                    - `DataSourceNotionPage` - 类型，包括以下属性：
                        - `page_icon: null | { type: string | null; url: string | null; emoji: string | null }` - 页面图标（可选）
                        - `page_id: string` - 页面的唯一标识符
                        - `page_name: string` - 页面名称
                        - `parent_id: string` - 父页面的唯一标识符
                        - `type: string` - 页面类型
                        - `is_bound: boolean` - 是否绑定
- **后端交互**:
    - **URL**: `/notion/pre-import/pages: web/app/components/base/notion-page-selector/base.tsx line 30`
    - **方法**: `GET`

28. modifyDocMetadata

- **用途**: 用于修改特定文档的元数据。在前端更新文档的类型和元数据信息。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
  - `body: { doc_type: string; doc_metadata: Record<string, any> }` - 请求体，包括文档类型和元数据
- **返回值**:
  - **描述**: 返回一个 Promise，用于修改文档元数据，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/metadata`
  - **方法**: `PUT`

29. fetchSegments

- **用途**: 用于获取特定文档的段信息。在前端显示和管理文档的段落信息。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
  - `params: SegmentsQuery` - 请求参数，包括段查询条件
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取文档的段信息，解析后提供段的详细信息。
  - **数据类型**:
    - `SegmentsResponse` - 响应数据类型，包括以下属性：
      - `data: SegmentDetailModel[]` - 段列表，每个段为 `SegmentDetailModel` 类型
        - `SegmentDetailModel` - 类型，包括以下属性：
          - `id: string` - 段的唯一标识符
          - `position: number` - 段的位置
          - `document_id: string` - 文档的唯一标识符
          - `content: string` - 段的内容
          - `word_count: number` - 单词数
          - `tokens: number` - 令牌数
          - `keywords: string[]` - 关键词列表
          - `index_node_id: string` - 索引节点 ID
          - `index_node_hash: string` - 索引节点哈希
          - `hit_count: number` - 命中次数

                - `enabled: boolean` - 是否启用
                - `disabled_at: number` - 禁用时间
                - `disabled_by: string` - 禁用者
                - `status: SegmentStatus` - 段的状态
                - `created_by: string` - 创建者
                - `created_at: number` - 创建时间
                - `indexing_at: number` - 索引时间
                - `completed_at: number` - 完成时间
                - `error: string | null` - 错误信息
                - `stopped_at: number` - 停止时间
                - `answer?: string` - 答案（可选）
            - `has_more: boolean` - 是否有更多数据
            - `limit: number` - 每页数量
            - `total: number` - 段总数
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/documents/${documentId}/segments`
    - **方法**: `GET`

30. enableSegment

- **用途**: 用于启用特定段。在前端将特定段落状态设置为启用。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `segmentId: string` - 段的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于启用段，解析后提供操作结果。
    - **数据类型**:
        - `CommonResponse` - 响应数据类型，包括操作结果信息
            - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/segments/${segmentId}/enable`
    - **方法**: `PATCH`

31. disableSegment

- **用途**: 用于禁用特定段。在前端将特定段落状态设置为禁用。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `segmentId: string` - 段的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于禁用段，解析后提供操作结果。

    - **数据类型**:
        - `CommonResponse` - 响应数据类型，包括操作结果信息
            - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/segments/${segmentId}/disable`
    - **方法**: `PATCH`

32. updateSegment

- **用途**: 用于更新特定段的信息。在前端修改段的内容和相关信息。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `documentId: string` - 文档的唯一标识符
    - `segmentId: string` - 段的唯一标识符
    - `body: SegmentUpdator` - 请求体，包括段的更新内容
- **返回值**:
    - **描述**: 返回一个 Promise，用于更新段信息，解析后提供操作结果。
    - **数据类型**:
        - `{ data: SegmentDetailModel; doc_form: string }` - 响应数据类型，包括段的详细信息和文档形式
            - `SegmentDetailModel` - 类型，包括以下属性：
                - `id: string` - 段的唯一标识符
                - `position: number` - 段的位置
                - `document_id: string` - 文档的唯一标识符
                - `content: string` - 段的内容
                - `word_count: number` – 段的单词数
                - `tokens: number` - 令牌数
                - `keywords: string[]` - 关键词列表
                - `index_node_id: string` - 索引节点 ID
                - `index_node_hash: string` - 索引节点哈希
                - `hit_count: number` - 命中次数
                - `enabled: boolean` - 是否启用
                - `disabled_at: number` - 禁用时间
                - `disabled_by: string` - 禁用者
                - `status: SegmentStatus` - 段的状态
                - `created_by: string` - 创建者
                - `created_at: number` - 创建时间
                - `indexing_at: number` - 索引时间
                - `completed_at: number` - 完成时间
                - `error: string | null` - 错误信息
                - `stopped_at: number` - 停止时间
                - `answer?: string` - 答案（可选）

- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/segments/${segmentId}`
  - **方法**: `PATCH`

33. addSegment

- **用途**: 用于向特定文档添加新段。此函数帮助用户在前端向文档中添加新的段信息。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
  - `body: SegmentUpdator` - 请求体，包括段的内容和相关信息
- **返回值**:
  - **描述**: 返回一个 Promise，用于添加新段，解析后提供添加所段落的具体信息。
  - **数据类型**:
    - `{ data: SegmentDetailModel; doc_form: string }` - 响应数据类型，包括段的详细信息和文档形式
      - `SegmentDetailModel` - 同 `32. updateSegment`
      - 中的 `SegmentDetailModel` 类型
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/segment`
  - **方法**: `POST`

34. deleteSegment

- **用途**: 用于删除特定段。在前端移除不再需要的段落信息。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `documentId: string` - 文档的唯一标识符
  - `segmentId: string` - 段的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于删除段，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/documents/${documentId}/segments/${segmentId}`
  - **方法**: `DELETE`

35. segmentBatchImport

- **用途**: 用于批量导入段。在前端将多个段落信息批量导入到文档中。
- **属性**:
  - `url: string` - 请求的 URL
  - `body: FormData` - 请求体，包括批量导入的段信息
- **返回值**:
  - **描述**: 返回一个 Promise，用于批量导入段，解析后提供导入任务的详细信息。
  - **数据类型**:
    - `{ job_id: string; job_status: string }` - 响应数据类型，包括任务 ID 和任务状态
- **后端交互**:
  - **URL**: 动态生成
  - **方法**: `POST`

36. checkSegmentBatchImportProgress

- **用途**: 用于检查批量导入段的进度。在前端查看批量导入任务的进度。
- **属性**:
  - `jobID: string` - 批量导入任务的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于检查任务进度，解析后提供任务的详细信息。
  - **数据类型**:
    - `{ job_id: string; job_status: string }` - 响应数据类型，包括任务 ID 和任务状态
- **后端交互**:
  - **URL**: `/datasets/batch_import_status/${jobID}`
  - **方法**: `GET`

37. hitTesting

- **用途**: 用于在数据集上执行命中测试。在前端输入查询文本并获取与之匹配的段信息。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `queryText: string` - 查询文本
  - `retrieval_model: RetrievalConfig` - 检索模型配置，详细见 `web/types/app.ts line 376`
- **返回值**:
  - **描述**: 返回一个 Promise，用于执行命中测试，解析后提供测试结果。

- **数据类型**:
  - `HitTestingResponse` - 响应数据类型，包括以下属性：
    - `query: { content: string; tsne_position: TsnePosition }` - 查询信息，包括内容和 TSNE 位置
    - `records: Array<HitTesting>` - 命中记录列表
      - `HitTesting` - 类型，包括以下属性：
        - `segment: Segment` - 段信息
          - `Segment` - 类型，包括以下属性：
            - `id: string` - 段的唯一标识符
            - `document: Document` - 文档信息
              - `Document` - 类型，包括以下属性：
                - `id: string` - 文档的唯一标识符
                - `data_source_type: string` - 数据源类型
                - `name: string` - 文档名称
                - `doc_type: DocType` - 文档类型
            - `content: string` - 段的内容
            - `position: number` - 段的位置
            - `word_count: number` - 单词数
            - `tokens: number` - 令牌数
            - `keywords: string[]` - 关键词列表
            - `hit_count: number` - 命中次数
            - `index_node_hash: string` - 索引节点哈希
        - `score: number` - 匹配分数
        - `tsne_position: TsnePosition` - TSNE 位置
          - `TsnePosition` - 类型，包括以下属性：
            - `x: number` - x 坐标
            - `y: number` - y 坐标

- **后端交互**:
  - **URL**: `/datasets/${datasetId}/hit-testing`
  - **方法**: `POST`

38. fetchTestingRecords

- **用途**: 用于获取命中测试记录。在前端查看历史的命中测试记录。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
  - `params: { page: number; limit: number }` - 请求参数，包括页码和每页数量
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取命中测试记录，解析后提供记录的详细信息。
  - **数据类型**:
    - `HitTestingRecordsResponse` - 响应数据类型，包括以下属性：
      - `data: HitTestingRecord[]` - 测试记录列表
        - `HitTestingRecord` - 类型，包括以下属性：
          - `id: string` - 记录的唯一标识符
          - `content: string` - 记录内容
          - `source: 'app' | 'hit_testing' | 'plugin'` - 记录来源
          - `source_app_id: string` - 来源的唯一标识符
          - `created_by_role: 'account' | 'end_user'` - 创建者角色
          - `created_by: string` - 创建者
          - `created_at: number` - 创建时间
      - `has_more: boolean` - 是否有更多数据
      - `limit: number` - 每页数量
      - `total: number` - 记录总数
      - `page: number` - 当前页码
- **后端交互**:
  - **URL**: `/datasets/${datasetId}/queries`
  - **方法**: `GET`

39. fetchFileIndexingEstimate

- **用途**: 用于获取文件的索引估算信息。在前端显示文件的索引估算，包括预估的处理时间和费用。
- **属性**:
  - `body: IndexingEstimateParams` - 请求体，包括索引估算的参数
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取索引估算信息，解析后提供估算的详细信息。
  - **数据类型**:
    - `FileIndexingEstimateResponse` - 响应数据类型，包括以下属性：

        - `total_nodes: number` - 总节点数
        - `tokens: number` - 估算的令牌数
        - `total_price: number` - 总价格
        - `currency: string` - 货币类型
        - `total_segments: number` - 总段数
        - `preview: string[]` - 预览
        - `qa_preview?: QA[]` - 问答预览（可选）
- **后端交互**:
    - **URL**: `/datasets/indexing-estimate`
    - **方法**: `POST`

40. fetchNotionPagePreview

- **用途**: 用于获取 Notion 页面的预览信息。在前端显示 Notion 页面的内容预览。
- **属性**:
    - `workspaceID: string` - 工作区的唯一标识符
    - `pageID: string` - 页面的唯一标识符
    - `pageType: string` - 页面类型
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取 Notion 页面的预览信息，解析后提供页面内容。
    - **数据类型**:
        - `{ content: string }` - 响应数据类型，包括页面内容
- **后端交互**:
    - **URL**: `/notion/workspaces/${workspaceID}/pages/${pageID}/${pageType}/preview`
    - **方法**: `GET`

41. fetchApiKeysList

- **用途**: 用于获取 API 密钥列表。在前端显示和管理所有的 API 密钥。
- **属性**:
    - `url: string` - 请求的 URL
    - `params: Record<string, any>` - 请求参数
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取 API 密钥列表，解析后提供密钥的详细信息。
    - **数据类型**:
        - `ApikeysListResponse` - 响应数据类型，包括以下属性：
            - `data: ApikeyItemResponse[]` - API 密钥列表，每个密钥为 `ApikeyItemResponse` 类型
                - `ApikeyItemResponse` – API 密钥信息

        - Id: string
        - Token: string
        - Last_uesd_at: string
        - Created_at: string
- **后端交互**:
    - **URL**: {URL}
        - 用例：/datasets/api-keys (web/app/components/develop/secret-key/secret-key-modal.tsx line 51)
    - **方法**: GET

42. delApikey

- **用途**: 用于删除特定 API 密钥。在前端移除不再需要的 API 密钥。
- **属性**:
    - url: string - 请求的 URL
    - params: Record<string, any> - 请求参数
- **返回值**:
    - **描述**: 返回一个 Promise，用于删除 API 密钥，解析后提供操作结果。
    - **数据类型**:
        - CommonResponse - 响应数据类型，包括操作结果信息
            - result: 'success' | 'fail' - 请求的结果
- **后端交互**:
    - **URL**: {URL}
        - 用例：/datasets/api-keys/${delKeyID} (web/app/components/develop/secret-key/secret-key-modal.tsx line 79)
    - **方法**: DELETE

43. createApikey

- **用途**: 用于创建一个新的 API 密钥。在前端生成新的 API 密钥。
- **属性**:
    - url: string - 请求的 URL
    - body: Record<string, any> - 请求体，包括 API 密钥的相关信息
- **返回值**:
    - **描述**: 返回一个 Promise，用于创建 API 密钥，解析后提供新创建的密钥详细信息。
    - **数据类型**:
        - CreateApiKeyResponse - 响应数据类型，包括以下属性：
            - id: string - 密钥的唯一标识符
            - token: string - 密钥令牌
            - created_at: string - 创建时间

- **后端交互**:
  - **URL**: `{URL}`
    - 用例：`/datasets/api-keys`：`web/app/components/develop/secret-key/secret-key-modal.tsx line 87`
  - **方法**: `POST`

44. fetchDatasetApiBaseUrl

- **用途**: 用于获取数据集的 API 基础 URL。在前端显示和管理数据集的 API 基础 URL。
- **属性**:
  - `url: string` - 请求的 URL
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取数据集的 API 基础 URL，解析后提供 URL 详细信息。
  - **数据类型**:
    - `{ api_base_url: string }` - 响应数据类型，包括 API 基础 URL
- **后端交互**:
  - **URL**: `{URL}`
    - 用例：`/datasets/api-base-info`：`web/app/(commonLayout)/datasets/Container.tsx line 39`
  - **方法**: `GET`

45. fetchDataSources

- **用途**: 用于获取数据源列表。在前端显示和管理所有的数据源。
- **属性**: 无
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取数据源列表，解析后提供详细信息。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `api-key-auth/data-source`
  - **方法**: `GET`

46. createDataSourceApiKeyBinding

- **用途**: 用于创建数据源与 API 密钥的绑定。在前端管理数据源与 API 密钥的绑定关系。

- **属性**:
  - `body: Record<string, any>` - 请求体，包括数据源与 API 密钥绑定的相关信息
- **返回值**:
  - **描述**: 返回一个 Promise，用于创建绑定关系，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `api-key-auth/data-source/binding`
  - **方法**: `POST`

47. removeDataSourceApiKeyBinding

- **用途**: 用于删除数据源与 API 密钥的绑定。在前端移除不再需要的绑定关系。
- **属性**:
  - `id: string` - 绑定关系的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于删除绑定关系，解析后提供操作结果。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `api-key-auth/data-source/${id}`
  - **方法**: `DELETE`

48. createFirecrawlTask

- **用途**: 用于创建 Firecrawl 任务。在前端初始化和启动 Firecrawl 任务，以抓取网站内容。
- **属性**:
  - `body: Record<string, any>` - 请求体，包括 Firecrawl 任务的相关信息
- **返回值**:
  - **描述**: 返回一个 Promise，用于创建 Firecrawl 任务，解析后提供任务详细信息。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `website/crawl`
  - **方法**: `POST`

49. checkFirecrawlTaskStatus

- **用途**: 用于检查 Firecrawl 任务的状态。在前端查看 Firecrawl 任务的进度和状态。
- **属性**:
  - `jobId: string` - Firecrawl 任务的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于检查任务状态，解析后提供任务的详细信息。
  - **数据类型**:
    - `CommonResponse` - 响应数据类型，包括操作结果信息
      - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
  - **URL**: `website/crawl/status/${jobId}`
  - **方法**: `GET`

50. fetchSupportFileTypes

- **用途**: 用于获取支持的文件类型。在前端显示和管理支持的文件类型信息。
- **属性**:
  - `url: string` - 请求的 URL
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取支持的文件类型，解析后提供详细信息。
  - **数据类型**:
    - `FileTypesRes` - 响应数据类型，包括以下属性：
      - `allowed_extensions: string[]` - 允许的文件扩展名
- **后端交互**:
  - **URL**: `{URL}`
    - 用例：`/files/support-type: web/app/components/datasets/create/file-uploader/index.tsx line 49`
  - **方法**: `GET`

51. getErrorDocs

- **用途**: 用于获取出错的文档信息。在前端显示和管理出错的文档。
- **属性**:
  - `datasetId: string` - 数据集的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取出错的文档信息，解析后提供详细信息。
  - **数据类型**:

        - `ErrorDocsResponse` - 响应数据类型，包括以下属性：
            - `data: IndexingStatusResponse[]` - 出错文档的索引状态列表
                - `IndexingStatusResponse` - 类型，包括以下属性：
                    - `id: string` - 文档的唯一标识符
                    - `indexing_status: DocumentIndexingStatus` - 索引状态
                        - 可能值有 `waiting, parsing, cleaning, splitting, indexing, paused, error, completed`
                    - `processing_started_at: number` - 处理开始时间
                    - `parsing_completed_at: number` - 解析完成时间
                    - `cleaning_completed_at: number` - 清理完成时间
                    - `splitting_completed_at: number` - 分割完成时间
                    - `completed_at: any` - 完成时间
                    - `paused_at: any` - 暂停时间
                    - `error: any` - 错误信息
                    - `stopped_at: any` - 停止时间
                    - `completed_segments: number` - 完成的段数
                    - `total_segments: number` - 总段数
            - `total: number` - 出错文档总数
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/error-docs`
    - **方法**: `GET`

52. retryErrorDocs

- **用途**: 用于重试出错的文档索引。在前端重新处理出错的文档。
- **属性**:
    - `datasetId: string` - 数据集的唯一标识符
    - `document_ids: string[]` - 出错文档的唯一标识符列表
- **返回值**:
    - **描述**: 返回一个 Promise，用于重试出错文档的索引，解析后提供操作结果。
    - **数据类型**:
        - `CommonResponse` - 响应数据类型，包括操作结果信息
            - `result: 'success' | 'fail'` - 请求的结果
- **后端交互**:
    - **URL**: `/datasets/${datasetId}/retry`
    - **方法**: `POST`

Web/service/debug.ts – *聊天和消息处理相关的 API 接口*

1. sendChatMessage

- **用途**: 用于发送聊天消息并处理服务器响应。在前端发送聊天消息并以流模式接收响应。
- **属性**:
  - `appId: string` - 应用的唯一标识符
  - `body: Record<string, any>` - 请求体，包括聊天消息的内容
  - `callbacks: { onData: IOnData; onCompleted: IOnCompleted; onThought: IOnThought; onFile: IOnFile; onError: IOnError; getAbortController?: (abortController: AbortController) => void; onMessageEnd: IOnMessageEnd; onMessageReplace: IOnMessageReplace }` - 回调函数对象，包括处理数据、完成、思考、文件、错误、中止控制器、消息结束和消息替换的回调, 详细见 `web/service/base.ts line 45-62`
- **返回值**:
  - **描述**: 返回一个 ssePost 方法，用于发送聊天消息并处理服务器响应。
  - **数据类型**:
    - `void` - 不返回具体数据，仅用于处理流响应
- **后端交互**:
  - **URL**: `apps/${appId}/chat-messages`
  - **方法**: `SSE POST`

2. stopChatMessageResponding

- **用途**: 用于停止特定聊天消息的响应。在前端停止正在进行的聊天消息响应。
- **属性**:
  - `appId: string` - 应用的唯一标识符
  - `taskId: string` – 聊天任务的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于停止聊天消息响应，解析后提供操作结果。
  - **数据类型**:
    - `void` - 不返回具体数据，仅用于停止响应
- **后端交互**:
  - **URL**: `apps/${appId}/chat-messages/${taskId}/stop`
  - **方法**: `POST`

3. sendCompletionMessage

- **用途**: 用于发送完成消息并处理服务器响应。在前端发送完成消息并以流模式接收响应。

- **属性**:
  - `appId: string` - 应用的唯一标识符
  - `body: Record<string, any>` - 请求体，包括完成消息的内容
  - `callbacks: { onData: IOnData; onCompleted: IOnCompleted; onError: IOnError; onMessageReplace: IOnMessageReplace }` - 回调函数对象，包括处理数据、完成、错误和消息替换的回调, 详细见 `web/service/base.ts line 45-62`
- **返回值**:
  - **描述**: 返回一个 Promise，用于发送完成消息并处理服务器响应，解析后提供响应的详细信息。
  - **数据类型**:
    - `void` - 不返回具体数据，仅用于处理流响应
- **后端交互**:
  - **URL**: `apps/${appId}/completion-messages`
  - **方法**: `SSE POST`

4. fetchSuggestedQuestions

- **用途**: 用于获取建议的问题。在前端显示和管理特定聊天消息的建议问题。
- **属性**:
  - `appId: string` - 应用的唯一标识符
  - `messageId: string` - 消息的唯一标识符
  - `getAbortController?: any` - 可选，中止控制器
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取建议问题，解析后提供问题的详细信息。
  - **数据类型**:
    - `any` - 返回的建议问题的数据类型
    - 用例见 `web/app/components/app/configuration/debug/debug-with-single-model/index.tsx line 98`
- **后端交互**:
  - **URL**: `apps/${appId}/chat-messages/${messageId}/suggested-questions`
  - **方法**: `GET`

5. fetchConversationMessages

- **用途**: 用于获取会话消息。在前端显示和管理特定会话的所有消息。
- **属性**:
  - `appId: string` - 应用的唯一标识符
  - `conversation_id: string` - 会话的唯一标识符
  - `getAbortController?: any` - 可选，中止控制器

- **返回值**:
  - **描述**: 返回一个 Promise，用于获取会话消息，解析后提供消息的详细信息。
  - **数据类型**:
    - `any` - 返回的会话消息的数据类型
- **后端交互**:
  - **URL**: `apps/${appId}/chat-messages`
  - **方法**: `GET`

6. generateRule

- **用途**: 用于生成规则。在前端根据提供的输入数据生成特定规则。
- **属性**:
  - `body: Record<string, any>` - 请求体，包括生成规则所需的相关信息
- **返回值**:
  - **描述**: 返回一个 Promise，用于生成规则，解析后提供生成的规则详细信息。
  - **数据类型**:
    - `AutomaticRes` - 响应数据类型，包括以下属性：
      - `prompt: string` - 提示文本
      - `variables: string[]` - 变量列表
      - `opening_statement: string` - 开场声明
- **后端交互**:
  - **URL**: `/rule-generate`
  - **方法**: `POST`

7. fetchModelParams

- **用途**: 用于获取指定模型的参数规则。在前端显示和管理不同模型提供者提供的模型参数设置，以便用户了解模型的可配置选项及其详细信息。
- **属性**:
  - `providerName: string` - 模型提供者的名称
  - `modelId: string` - 模型的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取模型参数规则，解析后提供参数的详细信息。
  - **数据类型**:
    - `{ data: ModelParameterRule[] }` - 响应数据类型，包括模型参数规则列表
      - `ModelParameterRule` - 类型，包括以下属性：
        - `default?: number | string | boolean | string[]` - 参数默认值（可选）

        - `help?: TypeWithI18N` - 参数帮助信息（可选）
        - `label: TypeWithI18N` - 参数标签
            - `TypeWithI18N` 详细见 `web/app/components/header/account-setting/model-provider-page/declarations.ts line 3`
        - `min?: number` - 最小值（可选）
        - `max?: number` - 最大值（可选）
        - `name: string` - 参数名称
        - `precision?: number` - 精度（可选）
        - `required: false` - 是否必需
        - `type: string` - 参数类型
        - `use_template?: string` - 使用模板（可选）
        - `options?: string[]` - 参数选项（可选）
        - `tagPlaceholder?: TypeWithI18N` - 标签占位符（可选）
- **后端交互**:
    - **URL**: `/workspaces/current/model-providers/${providerName}/models/parameter-rules`
    - **方法**: `GET`

8. fetchPromptTemplate

- **用途**: 用于获取提示模板。在前端显示和管理特定模式下的提示模板，用户可以根据应用模式、模型模式和是否设置了数据集来获取合适的提示模板配置。
- **属性**:
    - `params: {`
        - `appMode: string` - 应用模式
        - `mode: ModelModeType` - 模型模式
            - `ModelModeType`
                - `chat = 'chat',`
                - `completion = 'completion',`
                - `unset = '',`
        - `modelName: string` - 模型名称
        - `hasSetDataSet: boolean` - 数据集设置标识
    - `}` - 请求参数
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取提示模板，解析后提供模板的详细信息。
    - **数据类型**:
        - `Object` - 响应数据类型
            - `chat_prompt_config: ChatPromptConfig` - 聊天提示配置
                - `ChatPromptConfig` - 类型，包括以下属性：
                    - `prompt: PromptItem[]` – 提示项列表

                - `PromptItem` 类型，包括以下属性：
                    - `role?: PromptRole` - 提示角色（可选）
                    - `text: string` - 提示文本
            - `completion_prompt_config: CompletionPromptConfig` - 完成提示配置
                - `CompletionPromptConfig` - 类型，包括以下属性：
                    - `prompt: PromptItem` - 提示文本
                    - `conversation_histories_role: ConversationHistoriesRole` - 会话历史角色
                    - `ConversationHistoriesRole` 类型，包括以下属性：
                        - `user_prefix: string` - 用户前缀
                        - `assistant_prefix: string` – 协助助理前缀
            - `stop: []` - 停止标识
- **后端交互**:
    - **URL**: `/app/prompt-templates`
    - **方法**: `GET`

9. fetchTextGenerationMessage

- **用途**: 用于获取特定消息的文本生成内容。在前端显示和管理指定消息的详细生成内容，用于展示或进一步处理。
- **属性**:
    - `params: { appId: string; messageId: string }` - 请求参数，包括应用的唯一标识符和消息的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取文本生成消息，解析后提供消息的详细信息。
    - **数据类型**:
        - `any` - 返回的文本生成消息的数据类型
        - 用例见 `web/app/components/app/text-generate/item/index.tsx line 203`
- **后端交互**:
    - **URL**: `/apps/${appId}/messages/${messageId}`
    - **方法**: `GET`

Web/service/explore.ts – “探索” (Explore)*页面相关的 API 接口*

1. fetchAppList

- **用途**: 用于获取应用列表，包括应用的分类和推荐的应用。
- **属性**: 无
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取应用列表，解析后提供分类和推荐的应用信息。
  - **数据类型**:
    - `{ categories: AppCategory[]; recommended_apps: App[] }` - 响应数据类型，包括应用分类和推荐的应用
      - `AppCategory` - 类型，为预定义的字符串类型
        - 可能值：`'Writing', 'Translate', 'HR', 'Programming', 'Assistant'`
      - `App` - 类型，包括以下属性：
        - `app: AppBasicInfo` - 应用的基本信息
          - `AppBasicInfo` - 类型，包括以下属性：
            - `id: string` - 应用的唯一标识符
            - `mode: AppMode` - 应用模式
              - 可能值：`'advanced-chat', 'agent-chat', 'chat', 'completion', 'workflow'`
            - `icon: string` - 应用图标
            - `icon_background: string` - 应用图标背景
            - `name: string` - 应用名称
            - `description: string` - 应用描述
        - `app_id: string` - 应用 ID
        - `description: string` - 应用描述
        - `copyright: string` - 版权信息
        - `privacy_policy: string | null` - 隐私政策（可选）
        - `custom_disclaimer: string | null` - 自定义免责声明（可选）
        - `category: AppCategory` - 应用分类
        - `position: number` - 应用位置
        - `is_listed: boolean` - 是否列出
        - `install_count: number` - 安装次数
        - `installed: boolean` - 是否已安装
        - `editable: boolean` - 是否可编辑
        - `is_agent: boolean` - 是否为代理

- **后端交互**:
  - **URL**: `/explore/apps`
  - **方法**: `GET`

2. fetchAppDetail

- **用途**: 用于获取特定应用的详细信息。
- **属性**:
  - `id: string` - 应用的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取应用详情。
  - **数据类型**:
    - `Promise<any>` - 返回的应用详情的数据类型（`string?`）
    - 用例：`web/app/components/explore/app-list/index.tsx line 100 -> 105, importApp()` accepts `data` as `string`
- **后端交互**:
  - **URL**: `/explore/apps/${id}`
  - **方法**: `GET`

3. fetchInstalledAppList

- **用途**: 用于获取已安装的应用列表。
- **属性**: 无
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取已安装的应用列表。
  - **数据类型**:
    - `Promise<any>` - 返回的已安装应用列表的数据类型
    - 用例：`web/app/components/explore/sidebar/index.tsx line 58`
- **后端交互**:
  - **URL**: `/installed-apps`
  - **方法**: `GET`

4. installApp

- **用途**: 用于将特定应用安装到用户的工作空间中。
- **属性**:
  - `id: string` - 应用的唯一标识符
- **返回值**:
  - **描述**: 返回一个 Promise，用于安装应用，解析后提供操作结果。
  - **数据类型**:
    - `Promise<any>` - 返回的操作结果的数据类型
- **后端交互**:
  - **URL**: `/installed-apps`

    - **方法**: `POST`
    - **请求体**:
        - `body: { app_id: id }` - 请求体，包括应用的唯一标识符

5. uninstallApp

- **用途**: 用于从用户的工作空间中卸载特定应用。
- **属性**:
    - `id: string` - 应用的唯一标识符
- **返回值**:
    - **描述**: 返回一个 Promise，用于卸载应用。
    - **数据类型**:
        - `Promise<any>` - 返回的操作结果的数据类型
        - 用例：`web/app/components/explore/sidebar/index.tsx line 66`
- **后端交互**:
    - **URL**: `/installed-apps/${id}`
    - **方法**: `DELETE`

6. updatePinStatus

- **用途**: 用于更改特定应用在用户界面中的固定（置顶）状态。
- **属性**:
    - `id: string` - 应用的唯一标识符
    - `isPinned: boolean` - 应用是否被置顶
- **返回值**:
    - **描述**: 返回一个 Promise，用于更新应用的置顶状态。
    - **数据类型**:
        - `Promise<any>` - 返回的操作结果的数据类型
        - 用例： `web/app/components/explore/sidebar/index.tsx line 76`
- **后端交互**:
    - **URL**: `/installed-apps/${id}`
    - **方法**: `PATCH`
    - **请求体**:
        - `body: { is_pinned: isPinned }` - 请求体，包括应用的置顶状态

7. getToolProviders

- **用途**: 用于获取当前工作空间中的工具提供者列表。
- **属性**: 无
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取工具提供者列表。
    - **数据类型**:

        - `Promise<any>` - 返回的工具提供者列表的数据类型
- **后端交互**:
    - **URL**: `/workspaces/current/tool-providers`
    - **方法**: `GET`

Web/service/log.ts – *获取和操作日志、会话、消息和工作流相关数据的 API 接口*

会话管理模块

主要用于获取不同类型应用的会话列表和详细信息。这些端点帮助用户在前端界面上查看和管理会话记录，了解每个会话的详细信息，包括消息、反馈、注释等。

1. fetchConversationList

- **用途**:用于获取特定应用中所有会话的列表，包括每个会话的基本信息，如会话 ID、问题、回答、用户评分等。
- **属性**:
    - `appId: string` - 应用的唯一标识符
    - `params?: Record<string, any>` - 可选的请求参数
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取会话列表，解析后提供会话的详细信息。
    - **数据类型**:
        - `ConversationListResponse` - 类型，包括以下属性：
            - `logs: Conversation[]` - 会话列表
                - `Conversation` - 类型，包括以下属性：
                    - `id: string` - 会话的唯一标识符
                    - `key: string` - 会话的 key
                    - `conversationId: string` - 会话 ID
                    - `question: string` - 会话中的问题
                    - `answer: string` - 会话中的回答
                    - `userRate: number` - 用户评分
                    - `adminRate: number` - 管理员评分
- **后端交互**:
    - **URL**: `/console/api/apps/${appId}/messages`
    - **方法**: `GET`

2. fetchCompletionConversations

- **用途**:用于获取**文本生成应用**中所有会话的列表，包括每个会话的基本信息，如会话 ID、状态、来源、用户反馈统计等。
- **属性**:
  - `url: string` - 请求的 URL
  - `params?: CompletionConversationsRequest` - 可选的请求参数
- **返回值**:
  - **描述**: 返回一个 Promise，用于获取文本生成应用的会话列表，解析后提供会话的详细信息。
  - **数据类型**:
    - `CompletionConversationsResponse` - 类型，包括以下属性：
      - `data: CompletionConversationGeneralDetail[]` - 会话列表
        - `CompletionConversationGeneralDetail` - 类型，包括以下属性：
          - `id: string` - 会话的唯一标识符
          - `status: 'normal' | 'finished'` - 会话状态
          - `from_source: 'api' | 'console'` - 会话来源
          - `from_end_user_id: string` - 用户 ID
          - `from_end_user_session_id: string` - 用户会话 ID
          - `from_account_id: string` - 账户 ID
          - `read_at: Date` - 阅读时间
          - `created_at: number` - 创建时间
          - `annotation: Annotation` - 注释信息
            - `Annotation` - 类型, 详见 `web/models/log.ts line 70`
          - `user_feedback_stats: { like: number; dislike: number }` - 用户反馈统计
          - `admin_feedback_stats: { like: number; dislike: number }` - 管理员反馈统计
          - `model_config: { provider: string; model_id: string; configs: Pick<ModelConfigDetail, 'prompt_template'> }` - 模型配置
            - `ModelConfigDetail` - 类型详见 `web/models/log.ts line 45`
          - `message: Pick<MessageContent, 'inputs' | 'query' | 'answer' | 'message'>` - 消息内容
            - `MessageContent` - 类型详见 `web/models/log.ts line 77`
      - `has_more: boolean` - 是否还有更多数据
      - `limit: number` - 每页的限制数量
      - `total: number` - 数据总量

        - `page: number` - 当前页码
- **后端交互**:
    - **URL**: `{url}`
        - 用例：`/apps/${appDetail.id}/completion-conversations (web/app/components/app/log/index.tsx line 89)`
    - **方法**: `GET`

3. fetchCompletionConversationDetail

- **用途**:用于获取**文本生成应用**中某个会话的详细信息，包括模型配置、消息内容等。
- **属性**:
    - `url: string` - 请求的 URL
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取文本生成应用的会话详情，解析后提供会话的详细信息。
    - **数据类型**:
        - `CompletionConversationFullDetailResponse` - 类型，包括以下属性：
            - `id: string` - 会话的唯一标识符
            - `status: 'normal' | 'finished'` - 会话状态
            - `from_source: 'api' | 'console'` - 会话来源
            - `from_end_user_id: string` - 用户 ID
            - `from_account_id: string` - 账户 ID
            - `created_at: number` - 创建时间
            - `model_config: { provider: string; model_id: string; configs: ModelConfigDetail }` - 模型配置
                - `ModelConfigDetail` - 类型详见 `web/models/log.ts line 45`
            - `message: MessageContent` - 消息内容
                - `MessageContent` - 类型，包括以下属性：
                    - `id: string` - 消息的唯一标识符
                    - `conversation_id: string` - 会话 ID
                    - `query: string` - 查询
                    - `inputs: Record<string, any>` - 输入
                    - `message: { role: string; text: string; files?: VisionFile[] }[]` - 消息列表
                    - `message_tokens: number` - 消息令牌数
                    - `answer_tokens: number` - 回答令牌数
                    - `answer: string` - 回答
                    - `provider_response_latency: number` - 提供者响应延迟
                    - `created_at: number` - 创建时间

        - `annotation: LogAnnotation` - 注释
            - `LogAnnotation` - 类型详见 `web/models/log.ts line 59`
        - `annotation_hit_history: { annotation_id: string; annotation_create_account: { id: string; name: string; email: string }; created_at: number }` - 注释命中历史
        - `feedbacks: Array<{ rating: 'like' | 'dislike' | null; content: string | null; from_source?: 'admin' | 'user'; from_end_user_id?: string }>` - 反馈
        - `message_files: VisionFile[]` - 消息文件
        - `agent_thoughts: any[]` - 代理想法
        - `workflow_run_id: string` - 工作流运行 ID
- **后端交互**:
    - **URL**: `{url}`
        - 用例：`/apps/${appId}/completion-conversations/${conversationId}` (web/app/components/app/log/list.tsx line 541)
    - **方法**: `GET`

4. fetchChatConversations

- **用途**:用于获取**聊天应用**中所有会话的列表，包括每个会话的基本信息，如会话 ID、状态、来源、用户反馈统计等。
- **属性**:
    - `url: string` - 请求的 URL
    - `params?: ChatConversationsRequest` - 可选的请求参数
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取聊天应用的会话列表，解析后提供会话的详细信息。
    - **数据类型**:
        - `ChatConversationsResponse` - 类型，包括以下属性：
            - `data: ChatConversationGeneralDetail[]` - 会话列表
                - `ChatConversationGeneralDetail` - 类型，包括以下属性：
                    - `id: string` - 会话的唯一标识符
                    - `status: 'normal' | 'finished'` - 会话状态
                    - `from_source: 'api' | 'console'` - 会话来源
                    - `from_end_user_id: string` - 用户 ID
                    - `from_end_user_session_id: string` - 用户会话 ID
                    - `from_account_id: string` - 账户 ID
                    - `read_at: Date` - 阅读时间

        - `created_at: number` - 创建时间
        - `summary: string` - 会话总结
        - `message_count: number` - 消息数量
        - `annotated: boolean` - 是否有注释
        - `user_feedback_stats: { like: number; dislike: number }` - 用户反馈统计
        - `admin_feedback_stats: { like: number; dislike: number }` - 管理员反馈统计
        - `model_config: { provider: string; model_id: string; configs: Pick<ModelConfigDetail, 'inputs' | 'query' | 'answer' | 'message'> }` - 模型配置
            - `ModelConfigDetail` - 类型详见 `web/models/log.ts line 45`

- **后端交互**:
    - **URL**: `{url}`
        - 用例：`/apps/${appID}/chat-conversations (web/app/components/app/log/index.tsx line 82)`
    - **方法**: `GET`

5. fetchChatConversationDetail

- **用途**:用于获取**聊天应用**中某个会话的详细信息，包括模型配置、消息内容等。
- **属性**:
    - `url: string` - 请求的 URL
- **返回值**:
    - **描述**: 返回一个 Promise，用于获取聊天应用的会话详情，解析后提供会话的详细信息。
    - **数据类型**:
        - `ChatConversationFullDetailResponse` - 类型，包括以下属性：
            - `Omit<CompletionConversationGeneralDetail, 'message' | 'model_config'>`，详见 `web/models/log.ts line 109`
            - `message_count: number` – 消息数量
            - `model_config: {` – 模型参数
              `provider: string`
              `model_id: string`
              `configs: ModelConfigDetail`
              `model: LogModelConfig`
              `}`

- **后端交互**:
    - **URL**: `{url}`
        - 用例：`/apps/${appId}/chat-conversations/${conversationId} (web/app/components/app/log/list.tsx line 586)`
    - **方法**: `GET`

消息管理模块

主要用于获取和更新特定会话中的消息列表和反馈信息。

6. fetchChatMessages
    - **用途**: 获取**聊天应用**中某个会话的消息列表。
    - **返回值**: `ChatMessagesResponse`
        - `data: Array<ChatMessage>`
            - `ChatMessage` 详见 `web/models/log.ts line 199 -> 77`
        - `has_more: boolean`
        - `limit: number`
    - **方法**: `GET`
    - **URL**: `{url}`
        - 用例：`/apps/${appDetail?.id}/chat-messages` `(web/app/components/app/log/list.tsx line 190)`

7. updateLogMessageFeedbacks
    - **用途**: 更新日志消息的反馈信息。
    - **返回值**: `LogMessageFeedbacksResponse`
        - `result: 'success' | 'error'` - 反馈结果
    - **方法**: `POST`
    - **URL**: `{url}`
        - 用例：`/apps/${appId}/feedbacks` `(web/app/components/app/log/list.tsx line 190)`

8. updateLogMessageAnnotations
    - **用途**: 更新日志消息的注释信息。
    - **返回值**: `LogMessageAnnotationsResponse`
        - `result: 'success' | 'error'` - 反馈结果
    - **方法**: `POST`
    - **URL**: `{url}`
        - 用例：`/apps/${appId}/annotations` `(web/app/components/app/log/list.tsx line 605)`

9. fetchAnnotationsCount
    - **用途**: 获取注释的统计信息。
    - **返回值**: `AnnotationsCountResponse`
        - `count: number`
    - **方法**: `GET`
    - **URL**: `{url}`
        - 用例：`/apps/${appId}/annotations/count` `(web/app/components/app/annotation/filter.tsx line 29)`

工作流日志管理模块

该模块主要用于获取和管理工作流的日志和运行详细信息。

10. fetchWorkflowLogs
    - **用途**: 获取工作流的日志列表。
    - **返回值**: `WorkflowLogsResponse`
      - `data: Array<WorkflowAppLogDetail>`
        - 详见 `dify/web/models/log.ts line 251`
      - `has_more: boolean`
      - `limit: number`
      - `total: number`
      - `page: number`
    - **方法**: `GET`
    - **URL**: `{url}`
      - 用例：`/apps/${appID}/workflow-app-logs` `(web/app/components/app/workflow-log/index.tsx line 69)`

11. fetchRunDetail
    - **用途**: 获取工作流运行的详细信息。
    - **返回值**: `WorkflowRunDetailResponse`
      - 详见 `web/models/log.ts line 275`
    - **方法**: `GET`
    - **URL**: `/apps/${appID}/workflow-runs/${runID}`

代理日志管理模块

该模块主要用于获取代理日志的详细信息和节点跟踪列表。

12. fetchTracingList
    - **用途**: 获取节点跟踪列表。
    - **返回值**: `NodeTracingListResponse`
      - 详见 `web/models/log.ts line 61`
    - **方法**: `GET`
    - **URL**: `/apps/${appID}/workflow-runs/${runID}/node-executions`

13. fetchAgentLogDetail
    - **用途**: 获取代理日志的详细信息。
    - **返回值**: `AgentLogDetailResponse`
      - 详见 `web/models/log.ts line 346`
    - **方法**: `GET`
    - **URL**: `/apps/${appID}/agent/logs`

Web/service/share.ts – 聊天应用和工作流相关的数据交互 API 接口

消息发送与处理模块

该模块主要用于发送和处理聊天消息、完成消息和工作流消息。

1. sendChatMessage
    - **用途**: 发送聊天消息，并处理消息流。
    - **属性**:
        - `body: Record<string, any>`
        - `callbacks`
        - `isInstalledApp: boolean`
        - `installedAppId: string`
    - **返回值**: `Promise<any>`,用于消息发送
    - **方法**: `ssePost`
2. stopChatMessageResponding
    - **用途**: 停止聊天消息响应。
    - **属性**:
        - `appId: string`
        - `taskId: string`
        - `isInstalledApp: boolean`
        - `installedAppId: string`
    - **返回值**: `Promise<any>`,用于停止消息回复
    - **方法**: `post`
3. sendCompletionMessage
    - **用途**: 发送完成消息，并处理消息流。
    - **属性**:
        - `body: Record<string, any>`
        - `callbacks`
        - `isInstalledApp: boolean`
        - `installedAppId: string`
    - **返回值**: `Promise<any>`
    - **方法**: `ssePost`
4. sendWorkflowMessage
    - **用途**: 发送工作流消息，并处理消息流。
    - **属性**:
        - `body: Record<string, any>`
        - `callbacks`
        - `isInstalledApp: boolean`
        - `installedAppId: string`

- **返回值**: `Promise<any>`
- **方法**: `ssePost`

会话管理模块

该模块主要用于管理聊天和工作流的会话，包括获取、固定、解固定、删除和重命名会话。

5. fetchConversations
    - **用途**: 获取会话列表。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
        - `last_id?: string`
        - `pinned?: boolean`
        - `limit?: number`
    - **返回值**:`AppConversationData`
        - 详见 `web/models/share.ts line 38`
    - **方法**: `get`
6. pinConversation
    - **用途**: 置顶会话。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
        - `id: string` - 对话 ID
    - **返回值**: `Promise<any>`
    - **方法**: `patch`
7. unpinConversation
    - **用途**: 解除置顶会话。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
        - `id: string` - 对话 ID
    - **返回值**: `Promise<any>`
    - **方法**: `patch`

8. delConversation
    - **用途**: 删除会话。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
        - `id: string - 对话 ID`
    - **返回值**: `Promise<any>`
    - **方法**: `del`

9. renameConversation
    - **用途**: 重命名会话。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
        - `id: string - 对话 ID`
        - `name: string - 新名字`
    - **返回值**: `Promise<any>`
    - **方法**: `post`

10. generationConversationName
    - **用途**: 自动生成会话名称。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
        - `id: string - 对话 ID`
    - **返回值**: `ConversationItem`
        - `id: string`
        - `name: string`
        - `inputs: Record<string, any> | null`
        - `introduction: string`
    - **方法**: `post`

11. fetchChatList
    - **用途**: 获取聊天列表。
    - **属性**:
        - `conversationId: string`
        - `isInstalledApp: boolean`
        - `installedAppId: string`
    - **返回值**: `Promise<any>`
    - **方法**: `get`

系统与应用配置模块

该模块主要用于获取系统功能和应用的配置。

12. fetchAppInfo
    - **用途**: 获取应用信息。
    - **属性**: 无
    - **返回值**: `AppData`
      - 详见 `web/models/share.ts line 29`
    - **方法**: `get`

13. fetchAppParams
    - **用途**: 获取应用参数。
    - **属性**:
      - `isInstalledApp: boolean`
      - `installedAppId: string`
    - **返回值**: `ChatConfig`
      - **详见** `web/app/components/base/chat/types.ts line 46`
    - **方法**: `get`

14. fetchSystemFeatures
    - **用途**: 获取系统功能。
    - **属性**: 无
    - **返回值**: `SystemFeatures`
      - 详见 `web/types/feature.ts`
    - **方法**: `get`

15. fetchWebSAMLSSOUrl
    - **用途**: 获取 Web SAML SSO URL(允许用户使用单一身份验证会话访问多个独立的应用程序。通过 Web SAML SSO URL，用户可以在不重复输入用户名和密码的情况下，无缝访问多个应用程序)。
    - **属性**:
      - `appCode: string` - redirectUrl 的最后一段，URL 路径中的特定应用程序或资源。
      - `redirectUrl: string`
    - **返回值**: `{ url: string }`
    - **方法**: `get`

16. fetchWebOIDCSSOUrl

    - **用途:** 获取 Web OIDC SSO URL（允许用户通过第三方身份提供者（Identity Provider, IdP）进行身份验证，并获取用户的基本信息）。
    - **属性:**
        - `appCode: string` - redirectUrl 的最后一段，URL 路径中的特定应用程序或资源。
        - `redirectUrl: string`
    - **返回值:** `{ url: string }`
    - **方法:** `get`

17. fetchWebOAuth2SSOUrl

    - **用途**: 获取 Web OAuth2 SSO URL（允许第三方应用程序在资源所有者的许可下访问其资源）。
    - **属性**:
        - `appCode: string` - redirectUrl 的最后一段，URL 路径中的特定应用程序或资源。
        - `redirectUrl: string`
    - **返回值**: `{ url: string }`
    - **方法**: `get`

18. fetchAppMeta

    - **用途**: 获取应用元数据。
    - **属性**:
        - `isInstalledApp: boolean`
        - `installedAppId: string`
    - **返回值**: `AppMeta`
        - `tool_icons: Record<string, string>`
    - **方法**: `get`

反馈与注释模块

该模块主要用于更新和获取反馈与注释信息。

19. updateFeedback

- **用途**: 更新反馈信息。
- **属性**:
    - `url: string`
    - `body: Feedbacktype`
    - `isInstalledApp: boolean`
    - `installedAppId: string`
- **返回值**: `Promise<any>`

- **方法**: `post`

20. fetchMoreLikeThis

- **用途**: 获取相似消息。
- **属性**:
  - `messageId: string`
  - `isInstalledApp: boolean`
  - `installedAppId: string`
- **返回值**: `Promise<any>`
- **方法**: `get`

21. saveMessage

- **用途**: 保存消息。
- **属性**:
  - `messageId: string`
  - `isInstalledApp: boolean`
  - `installedAppId: string`
- **返回值**: `Promise<any>`
  - 用例： `web/app/components/share/text-generation/index.tsx line 110`
- **方法**: `post`

22. fetchSavedMessage

- **用途**: 获取已保存的消息。
- **属性**:
  - `isInstalledApp: boolean`
  - `installedAppId: string`
- **返回值**: `Promise<any>`
  - 用例： `web/app/components/share/text-generation/index.tsx line 106`
- **方法**: `get`

23. removeMessage

- **用途**: 删除消息。
- **属性**:
  - `messageId: string`
  - `isInstalledApp: boolean`
  - `installedAppId: string`
- **返回值**: `Promise<any>`
  - 用例： `web/app/components/share/text-generation/index.tsx line 115`
- **方法**: `del`

24. fetchSuggestedQuestions

- **用途**: 获取建议问题。
- **属性**:
  - `messageId: string`
  - `isInstalledApp: boolean`
  - `installedAppId: string`
- **返回值**: `Promise<any>`
  - 用例： `web/app/components/base/chat/chat-with-history/chat-wrapper.tsx line 78`
- **方法**: `get`

其他功能模块

该模块包含其他一些功能，如音频转文本、文本转音频、获取访问令牌等。

25. audioToText

- **用途**: 将音频转为文本。
- **属性**:
  - `url: string`
  - `isPublicAPI: boolean`
  - `body: FormData`
- **返回值**: `Promise<{ text: string }>`
  - 用例：`web/app/components/base/voice-input/index.tsx line 107`
- **方法**: `post`

26. textToAudio

- **用途**: 将文本转为音频。
- **属性**:
  - `url: string`
  - `isPublicAPI: boolean`
  - `body: FormData`
- **返回值**: `Promise<{ data: string }>`
  - 用例： `web/app/components/base/audio-btn/index.tsx line 64`
- **方法**: `post`

27. fetchAccessToken

- **用途**: 获取访问令牌。
- **属性**:
  - `appCode: string`
- **返回值**: `Promise<{ access_token: string }>`
  - 用例： `web/app/components/share/utils.ts line 15`
- **方法**: `get`

Web/service/sso.ts – 用户获取相应的 SSO URL 进行身份验证的 API 接口

1. getUserSAMLSSOUrl

- **用途**: 获取用户的 SAML SSO URL。
- **属性**: 无
- **返回值**: `Promise<{ url: string }>`
    - 用例：`web/app/signin/userSSOForm.tsx line 44`
- **方法**: `get`
- **URL**: `/enterprise/sso/saml/login`

2. getUserOIDCSSOUrl

- **用途**: 获取用户的 OIDC SSO URL。
- **属性**: 无
- **返回值**: `Promise<{ url: string; state: string }>`
    - 用例：`web/app/signin/userSSOForm.tsx line 51`
- **方法**: `get`
- **URL**: `/enterprise/sso/oidc/login`

3. getUserOAuth2SSOUrl

- **用途**: 获取用户的 OAuth2 SSO URL。
- **属性**: 无
- **返回值**: `Promise<{ url: string; state: string }>`
    - 用例：`web/app/signin/userSSOForm.tsx line 59`
- **方法**: `get`
- **URL**: `/enterprise/sso/oauth2/login`

Web/service/tag.ts – 标签（Tag）管理和操作的 API 接口

1. fetchTagList

- **用途**: 获取指定类型的标签列表。
- **属性**:
  - `type: string` - 标签类型
- **返回值**: `Promise<Tag[]>`
  - `Tag` 类型包括：
    - `id: string`
    - `name: string`
    - `type: string`
    - `binding_count: number` – 绑定的次数
  - 用例： `web/app/components/base/tag-management/index.tsx line 29`
- **方法**: `get`
- **URL**: `/tags`

2. createTag

- **用途**: 创建新的标签。
- **属性**:
  - `name: string` - 标签名称
  - `type: string` - 标签类型
- **返回值**: `Promise<Tag>`
  - `Tag` **类型见** `1.fetchTagList()`
  - 用例：`web/app/components/base/tag-management/index.tsx line 42`
- **方法**: `post`
- **URL**: `/tags`

3. updateTag

- **用途**: 更新现有标签的名称。
- **属性**:
  - `tagID: string` - 标签的唯一标识符
  - `name: string` - 新的标签名称
- **返回值**: `Promise<any>`
  - 用例：`web/app/components/base/tag-management/tag-item-editor.tsx line 58`
- **方法**: `patch`
- **URL**: `/tags/${tagID}`

4. deleteTag

- **用途**: 删除指定的标签。
- **属性**:
  - `tagID: string` - 标签的唯一标识符
- **返回值**: `Promise<any>`
  - 用例：`web/app/components/base/tag-management/tag-item-editor.tsx line 87`
- **方法**: `del`
- **URL**: `/tags/${tagID}`

5. bindTag

- **用途**: 将标签绑定到目标对象。
- **属性**:
  - `tagIDList: string[]` - 标签 ID 列表
  - `targetID: string` - 目标对象的 ID
  - `type: string` - 目标对象的类型
- **返回值**: `Promise<any>`
  - 用例：`web/app/components/base/tag-management/selector.tsx line 81`
- **方法**: `post`
- **URL**: `/tag-bindings/create`

6. unBindTag

- **用途**: 将标签从目标对象上解绑。
- **属性**:
  - `tagID: string` - 标签的唯一标识符
  - `targetID: string` - 目标对象的 ID
  - `type: string` - 目标对象的类型
- **返回值**: `Promise<any>`
  - 用例：`web/app/components/base/tag-management/selector.tsx line 90`
- **方法**: `post`
- **URL**: `/tag-bindings/remove`

Web/service/tools.ts – 管理和操作工具集合（Collection）和工具（Tools）的 API 接口

工具集合操作模块

用于操作和管理工具（Tools）集合，包括获取工具集合列表、获取特定类型工具的列表。

1. fetchCollectionList

- **用途**: 获取当前工作空间中可用的工具提供者集合列表。
- **属性**: 无
- **返回值**: `Promise<Collection[]>`
  - **`Collection`**
    - `id: string` - 工具集合的唯一标识符
    - `name: string` - 工具集合的名称
    - `author: string` - 工具集合的作者
    - `description: TypeWithI18N` - 工具集合的描述
    - `icon: string | Emoji` - 工具集合的图标，可以是字符串或表情符号
    - `label: TypeWithI18N` - 工具集合的标签
    - `type: CollectionType` - 工具集合的类型
    - `team_credentials: Record<string, any>` - 团队凭据
    - `is_team_authorization: boolean` - 是否为团队授权
    - `allow_delete: boolean` - 是否允许删除
    - `labels: string[]` - 标签列表
- **后端交互**:
  - **URL**: `/workspaces/current/tool-providers`
  - **方法**: GET

2. fetchBuiltInToolList

- **用途**: 获取指定集合名称中的内置工具列表。
- **属性**:
  - `collectionName: string` - 集合名称
- **返回值**: `Promise<Tool[]>`
  - **`Tool`**
    - `name: string` - 工具的名称
    - `author: string` - 工具的作者
    - `label: TypeWithI18N` - 工具的标签
    - `description: any` - 工具的描述
    - `parameters: ToolParameter[]` - 工具的参数列表
      - `name: string` - 参数名称

        - `label: TypeWithI18N` - 参数标签
        - `human_description: TypeWithI18N` - 参数的人类可读描述
        - `type: string` - 参数类型
        - `form: string` - 参数形式
        - `llm_description: string` - 参数的 LLM 描述
        - `required: boolean` - 是否必填
        - `default: string` - 默认值
        - `options?: { label: TypeWithI18N, value: string }[]` - 可选值列表
        - `min?: number` - 最小值
        - `max?: number` - 最大值
    - `labels: string[]` - 工具相关的标签列表
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/builtin/${collectionName}/tools`
    - **方法**: GET

3. fetchCustomToolList

- **用途**: 获取指定集合名称的自定义工具列表。
- **属性**:
    - `collectionName: string` - 集合名称
- **返回值**: `Promise<Tool[]>`, 见 `2. fetchBuiltInToolList()`
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/api/tools?provider=${collectionName}`
    - **方法**: GET

4. fetchModelToolList

- **用途**: 获取指定集合名称的模型工具列表。
- **属性**:
    - `collectionName: string` - 集合名称
- **返回值**: `Promise<Tool[]>`, 见 `2. fetchBuiltInToolList()`
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/model/tools?provider=${collectionName}`
    - **方法**: GET

5. fetchWorkflowToolList

- **用途**: 获取指定工作流应用 ID 的工具列表。

- **属性**:
  - `appID: string` - 应用 ID。
- **返回值**: `Promise<Tool[]>`, 见 `2. fetchBuiltInToolList()`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/workflow/tools?workflow_tool_id=${appID}`
  - **方法**: GET

6. fetchAllBuiltInTools

- **用途**: 获取所有内置工具。
- **属性**: 无
- **返回值**: `Promise<ToolWithProvider[]>`
  - **`ToolWithProvider`**
    - `Collection` – 详见 `1.fetchCollectionList()`
    - `tools: Tool` – 详见 `2. fetchBuiltInToolList()`
- **后端交互**:
  - **URL**: `/workspaces/current/tools/builtin`
  - **方法**: GET

7. fetchAllCustomTools

- **用途**: 获取所有自定义工具。
- **属性**: 无
- **返回值**: `Promise<ToolWithProvider[]>`
  - **`ToolWithProvider`**
    - `Collection` – 详见 `1.fetchCollectionList()`
    - `tools: Tool` – 详见 `2. fetchBuiltInToolList()`
- **后端交互**:
  - **URL**: `/workspaces/current/tools/api`
  - **方法**: GET

8. fetchAllWorkflowTools

- **用途**: 获取所有工作流工具。
- **属性**: 无
- **返回值**: `Promise<ToolWithProvider[]>`
  - **`ToolWithProvider`**
    - `Collection` – 详见 `1.fetchCollectionList()`
    - `tools: Tool` – 详见 `2. fetchBuiltInToolList()`
- **后端交互**:
  - **URL**: `/workspaces/current/tools/workflow`
  - **方法**: GET

内置工具操作模块

用于操作和管理内置工具（Built-in Tools），包括获取内置工具列表、更新和删除工具凭据等。

9. fetchBuiltInToolCredentialSchema

- **用途**: 获取指定集合名称的内置工具凭证模式，包括必要的参数和配置信息。
- **属性**:
  - `collectionName: string` - 指定的工具集合名称。
- **返回值**: `Promise<ToolCredential[]>`
  - **`ToolCredential`**
    - `name: string` - 凭证字段的名称。
    - `label: TypeWithI18N` - 凭证字段的标签。
    - `help: TypeWithI18N` - 凭证字段的帮助信息。
    - `placeholder: TypeWithI18N` - 凭证字段的占位符文本。
    - `type: string` - 凭证字段的类型。
    - `required: boolean` - 是否为必填字段。
    - `default: string` - 默认值。
    - `options?: { label: TypeWithI18N, value: string }[]` - 可选值列表。
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/builtin/${collectionName}/credentials_schema`
  - **方法**: GET

10. fetchBuiltInToolCredential

- **用途**: 获取指定集合名称的内置工具凭证。
- **属性**:
  - `collectionName: string` - 指定的工具集合名称。
- **返回值**: `Promise<ToolCredential[]>`,
  - 见 `9.fetchBuiltInToolCredentialSchema()`返回值类型
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/builtin/${collectionName}/credentials`
  - **方法**: GET

11. updateBuiltInToolCredential

- **用途**: 更新指定集合名称的内置工具凭证。

- **属性**:
  - `collectionName: string` - 指定的工具集合名称。
  - `credential: Record<string, any>` - 更新的凭证信息。
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/tools/add-tool-modal/index.tsx line 136`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/builtin/${collectionName}/update`
  - **方法**: POST

12. removeBuiltInToolCredential

- **用途**: 删除指定集合名称的内置工具凭证。
- **属性**:
  - `collectionName: string` - 指定的工具集合名称。
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/tools/add-tool-modal/index.tsx line 147`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/builtin/${collectionName}/delete`
  - **方法**: POST

自定义工具操作模块

用于操作和管理自定义工具（Custom Tools），包括获取自定义工具列表、创建和更新自定义工具集合等。

13. parseParamsSchema

- **用途**: 解析并转换参数模式。
- **属性**:
  - `schema: string` - 模式字符串
- **返回值**: `Promise<{ parameters_schema: CustomParamSchema[]; schema_type: string }>`
  - **`CustomParamSchema`**
    - `operation_id: string` - 操作 ID
    - `summary: string` - 摘要
    - `server_url: string` - 服务器 URL
    - `method: string` - 方法
    - `parameters: ParamItem[]` - 参数列表
      - **`ParamItem`**
        - `name: string` - 参数名称
        - `label: TypeWithI18N` - 参数标签

            - `human_description: TypeWithI18N` - 参数描述
            - `llm_description: string` - 参数 LLM 描述
            - `type: string` - 参数类型
            - `form: string` - 参数表单
            - `required: boolean` - 是否必需
            - `default: string` - 默认值
            - `min?: number` - 最小值
            - `max?: number` - 最大值
            - `options?: { label: TypeWithI18N, value: string }[]` - 可选项
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/api/schema`
    - **方法**: POST

14. fetchCustomCollection

- **用途**: 获取指定集合名称的自定义工具集合。
- **属性**:
    - `collectionName: string` - 集合名称
- **返回值**: `Promise<CustomCollectionBackend>`
    - **`CustomCollectionBackend`**
        - `provider: string` - 提供者名称
        - `original_provider?: string` - 原提供者名称
        - `credentials: Credential` - 凭证信息
            - **`Credential`**
                - `auth_type: AuthType` - 认证类型
                - `api_key_header?: string` - API 密钥头
                - `api_key_value?: string` - API 密钥值
                - `api_key_header_prefix?: AuthHeaderPrefix` - API 密钥头前缀
        - `icon: Emoji` - 图标
            - **`Emoji`**
                - `background: string` - 背景色
                - `content: string` - 内容
        - `schema_type: string` - 模式类型
        - `schema: string` - 模式
        - `privacy_policy: string` - 隐私政策
        - `custom_disclaimer: string` - 自定义免责声明
        - `tools?: ParamItem[]` - 工具列表
            - **`ParamItem`**
                - `name: string` - 参数名称
                - `label: TypeWithI18N` - 参数标签
                - `human_description: TypeWithI18N` - 参数描述

                - `llm_description: string` - 参数 LLM 描述
                - `type: string` - 参数类型
                - `form: string` - 参数表单
                - `required: boolean` - 是否必需
                - `default: string` - 默认值
                - `min?: number` - 最小值
                - `max?: number` - 最大值
                - `options?: { label: TypeWithI18N, value: string }[]` - 可选项
            - `id: string` - 工具集合 ID
            - `labels: string[]` - 标签列表
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/api/get?provider=${collectionName}`
    - **方法**: GET

15. createCustomCollection

- **用途**: 创建自定义工具集合。
- **属性**:
    - `collection: CustomCollectionBackend` - 包含创建信息的自定义集合对象。
        - 见 `14. fetchCustomCollection()`返回值
- **返回值**: `Promise<any>`
    - **用例：** `web/app/components/tools/add-tool-modal/index.tsx line 98`
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/api/add`
    - **方法**: POST

16. updateCustomCollection

- **用途**: 更新自定义工具集合。
- **属性**:
    - `collection: CustomCollectionBackend` - 包含更新信息的自定义集合对象。
        - 见 `14. fetchCustomCollection()`返回值
- **返回值**: `Promise<any>`
    - **用例：** `web/app/components/tools/provider/detail.tsx line 87`
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/api/update`
    - **方法**: POST

17. removeCustomCollection

- **用途**: 删除自定义工具集合。
- **属性**:
  - `collectionName: string` - 集合名称
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/tools/provider/detail.tsx line 96`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/api/delete`
  - **方法**: POST

18. importSchemaFromURL

- **用途**: 从 URL 导入模式。
- **属性**:
  - `url: string` - URL
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/tools/edit-custom-collection-modal/get-schema.tsx line 36`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/api/remote`
  - **方法**: GET

19. testAPIAvailable

- **用途**: 测试 API 的可用性。
- **属性**:
  - `payload: any` - 测试数据包
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/tools/edit-custom-collection-modal/test-api.tsx line 54`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/api/test/pre`
  - **方法**: POST

20. fetchLabelList

- **用途**: 获取标签列表。
- **属性**: 无
- **返回值**: `Promise<Label[]>`
  - **`Label`**
    - `name: string` - 标签名称
    - `id: string` - 标签 ID

- **后端交互**:
  - **URL**: `/workspaces/current/tool-labels`
  - **方法**: GET

工作流工具操作模块

用于操作和管理工作流工具，包括创建、更新和删除工作流工具提供者。

21. createWorkflowToolProvider

- **用途**: 创建工作流工具提供者。
- **属性**:
  - `payload: WorkflowToolProviderRequest & { workflow_app_id: string }`
    - **`WorkflowToolProviderRequest`**
      - `name: string` - 名称
      - `icon: Emoji` - 图标
        - **`Emoji`**
          - `background: string` - 背景色
          - `content: string` - 内容
      - `description: string` - 描述
      - `parameters: WorkflowToolProviderParameter[]` - 参数列表
        - **`WorkflowToolProviderParameter`**
          - `name: string` - 参数名称
          - `form: string` - 参数表单
          - `description: string` - 参数描述
          - `required?: boolean` - 是否必需
          - `type?: string` - 参数类型
      - `labels: string[]` - 标签列表
      - `privacy_policy: string` - 隐私政策
    - `workflow_app_id: string` - 工作流应用 ID
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/tools/workflow-tool/configure-button.tsx line 137`
- **后端交互**:
  - **URL**: `/workspaces/current/tool-provider/workflow/create`
  - **方法**: POST

22. saveWorkflowToolProvider

- **用途**: 保存工作流工具提供者。
- **属性**:
  - `payload: WorkflowToolProviderRequest & Partial<{ workflow_app_id: string, workflow_tool_id: string }>`
    - **`WorkflowToolProviderRequest`**

        - 详见 21. createWorkflowToolProvider()属性
      - workflow_app_id?: string - 工作流应用 ID（可选）
      - workflow_tool_id?: string - 工作流工具 ID（可选）
- **返回值**: Promise<any>
  - **用例：**web/app/components/tools/workflow-tool/configure-button.tsx line 157
- **后端交互**:
  - **URL**: /workspaces/current/tool-provider/workflow/update
  - **方法**: POST

23. fetchWorkflowToolDetailByAppID

- **用途**: 获取指定应用 ID 的工作流工具详细信息。
- **属性**:
  - appID: string - 应用 ID
- **返回值**: Promise<WorkflowToolProviderResponse>
  - **WorkflowToolProviderResponse**
    - workflow_app_id: string - 工作流应用 ID
    - workflow_tool_id: string - 工作流工具 ID
    - label: string - 标签
    - name: string - 名称
    - icon: Emoji - 图标
      - **Emoji**
        - background: string - 背景色
        - content: string - 内容
    - description: string - 描述
    - synced: boolean - 是否已同步
    - tool: Object - 工具详细信息
      - author: string - 作者
      - name: string - 名称
      - label: TypeWithI18N - 标签
      - description: TypeWithI18N - 描述
      - labels: string[] - 标签列表
      - parameters: ParamItem[] - 参数列表
        - **ParamItem**
          - name: string - 参数名称
          - label: TypeWithI18N - 参数标签
          - human_description: TypeWithI18N - 参数人为描述
          - llm_description: string - 参数 LLM 描述
          - type: string - 参数类型
          - form: string - 参数表单
          - required: boolean - 是否必需

        - `default: string` - 默认值
        - `min?: number` - 最小值
        - `max?: number` - 最大值
        - `options?: { label: TypeWithI18N, value: string }[]` - 可选项
    - `privacy_policy: string` - 隐私政策
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/workflow/get?workflow_app_id=${appID}`
    - **方法**: GET

24. fetchWorkflowToolDetail

- **用途**: 获取指定工具 ID 的工作流工具详细信息。
- **属性**:
    - `toolID: string` - 工具 ID
- **返回值**: `Promise<WorkflowToolProviderResponse>`
    - **`WorkflowToolProviderResponse`**
        - 详见 `23. fetchWorkflowToolDetailByAppID()` 返回值
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/workflow/get?workflow_tool_id=${toolID}`
    - **方法**: GET

25. deleteWorkflowTool

- **用途**: 删除指定工具 ID 的工作流工具。
- **属性**:
    - `toolID: string` - 工作流工具的唯一标识符。
- **返回值**: `Promise<any>`
    - **用例：** `web/app/components/tools/provider/detail.tsx line 140`
- **后端交互**:
    - **URL**: `/workspaces/current/tool-provider/workflow/delete`
    - **方法**: POST

Web/service/workflow.ts – 工作流（workflow）操作/管理相关的 API 接口

工作流草稿管理模块

用于操作和管理工作流草稿，包括获取、同步和发布工作流草稿等操作。

1. fetchWorkflowDraft

- **用途**: 获取工作流草稿。
- **属性**:
  - `url: string` - 请求的 URL。
- **返回值**: `Promise<FetchWorkflowDraftResponse>`
  - **`FetchWorkflowDraftResponse`**
    - `id: string` - 工作流草稿的唯一标识符。
    - `graph: object`
      - `nodes: Node[]` - 工作流图表中的节点数组。
        - `Node`,详见 `web/app/components/workflow/types.ts line 73`
      - `edges: Edge[]` - 工作流图表中的边数组。
        - `Edge`，详见 `web/app/components/workflow/types.ts line 80`
      - `viewport?: Viewport` - 视口信息（可选）。
    - `features?: any` - 工作流特性数据（可选）。
    - `created_at: number` - 创建时间。
    - `created_by: object`
      - `id: string` - 创建者的唯一标识符。
      - `name: string` - 创建者的名字。
      - `email: string` - 创建者的邮箱。
    - `hash: string` - 数据哈希值。
    - `updated_at: number` - 更新时间。
    - `tool_published: boolean` - 工具是否已发布。
- **后端交互**:
  - **URL**: `{URL}`
    - **用例：** `/apps/${appId}/workflows/draft`（`web/app/components/workflow/hooks/use-workflow-interactions.ts line 63`）
  - **方法**: GET

2. syncWorkflowDraft

- **用途**: 同步工作流草稿。

- **属性**:
  - `url: string` - 请求的 URL。
  - `params: Pick<FetchWorkflowDraftResponse, 'graph' | 'features'>` - 同步参数，包括工作流图表和特性数据。
- **返回值**: `Promise<CommonResponse & { updated_at: number; hash: string }>`
  - **`CommonResponse`**
    - `result: 'success' | 'fail'` - 请求的结果。
  - `updated_at: number` - 更新时间。
  - `hash: string` - 数据哈希值。
- **后端交互**:
  - **URL**: `{URL}`
    - **用例**：`/apps/${appId}/workflows/draft (web/app/components/workflow/hooks/use-workflow.ts line 482)`
  - **方法**: POST

3. fetchPublishedWorkflow

- **用途**: 获取发布的工作流。
- **属性**:
  - `url: string` - 请求的 URL。
- **返回值**: `Promise<FetchWorkflowDraftResponse>`
  - **`FetchWorkflowDraftResponse`**
    - `id: string` - 工作流草稿的唯一标识符。
    - `graph: object`
      - `nodes: Node[]` - 工作流图表中的节点数组。
      - `edges: Edge[]` - 工作流图表中的边数组。
      - `viewport?: Viewport` - 视口信息（可选）。
    - `features?: any` - 工作流特性数据（可选）。
    - `created_at: number` - 创建时间。
    - `created_by: object`
      - `id: string` - 创建者的唯一标识符。
      - `name: string` - 创建者的名字。
      - `email: string` - 创建者的邮箱。
    - `hash: string` - 数据哈希值。
    - `updated_at: number` - 更新时间。
    - `tool_published: boolean` - 工具是否已发布。
- **后端交互**:
  - **URL**: `{URL}`
    - **用例**：`/apps/${appID}/workflows/publish (web/app/components/workflow/hooks/use-workflow.ts line 509)`
  - **方法**: GET

4. publishWorkflow

- **用途**: 发布工作流。
- **属性**:
  - `url: string` - 请求的 URL。
- **返回值**: `Promise<CommonResponse & { created_at: number }>`
  - **`CommonResponse`**
    - `result: 'success' | 'fail'` - 请求的结果。
  - `created_at: number` - 创建时间。
- **后端交互**:
  - **URL**: `{URL}`
    - **用例：** `/apps/${appID}/workflows/publish` （`web/app/components/workflow/header/index.tsx line 113`）
  - **方法**: POST

节点配置和运行历史管理模块

用于获取节点的默认配置、运行历史记录以及单节点运行操作。

5. fetchNodesDefaultConfigs

- **用途**: 获取节点的默认配置。
- **属性**:
  - `url: string` - 请求的 URL。
- **返回值**: `Promise<NodesDefaultConfigsResponse>`
  - **`NodesDefaultConfigsResponse`**
    - `type: string` - 节点类型。
    - `config: any` - 节点配置。
- **后端交互**:
  - **URL**: `{URL}`
    - **用例：** `/apps/${appID}/workflows/default-workflow-block-configs` (`web/app/components/workflow/hooks/use-workflow.ts line 508`)
  - **方法**: GET

6. fetchWorkflowRunHistory

- **用途**: 获取工作流运行历史记录。
- **属性**:
  - `url: string` - 请求的 URL。

- **返回值**: `Promise<WorkflowRunHistoryResponse>`
  - **WorkflowRunHistoryResponse**
    - `data: WorkflowRunHistory[]` - 工作流运行历史记录数组。
      - **WorkflowRunHistory**
        - `id: string` - 工作流运行的唯一标识符。
        - `sequence_number: number` - 序列号。
        - `version: string` - 版本号。
        - `conversation_id?: string` - 对话 ID（可选）。
        - `message_id?: string` - 消息 ID（可选）。
        - `graph: object`
          - `nodes: Node[]` - 工作流图表中的节点数组。
          - `edges: Edge[]` - 工作流图表中的边数组。
          - `viewport?: Viewport` - 视口信息（可选）。
        - `inputs: Record<string, string>` - 输入参数。
        - `status: string` - 工作流状态。
        - `outputs: Record<string, any>` - 输出参数。
        - `error?: string` - 错误信息（可选）。
        - `elapsed_time: number` - 运行时间。
        - `total_tokens: number` - 总令牌数。
        - `total_steps: number` - 总步骤数。
        - `created_at: number` - 创建时间。
        - `finished_at: number` - 完成时间。
        - `created_by_account: object`
          - `id: string` - 创建者的唯一标识符。
          - `name: string` - 创建者的名字。
          - `email: string` - 创建者的邮箱。
- **后端交互**:
  - **URL**: `{URL}`
    - **用例：** `/apps/${appID}/workflow-runs` (web/app/components/workflow/header/view-history.tsx line 68)
  - **方法**: GET

7. fetchChatRunHistory

- **用途**: 获取聊天运行历史记录。
- **属性**:
  - `url: string` - 请求的 URL。
- **返回值**: `Promise<ChatRunHistoryResponse>`
  - **ChatRunHistoryResponse**
    - `data: WorkflowRunHistory[]` - 聊天运行历史记录数组。
      - **WorkflowRunHistory**
        - 详见 `6. fetchWorkflowRunHistory()`返回值

- **后端交互**:
  - **URL**: {URL}
    - **用例：** `/apps/${appID}/advanced-chat/workflow-runs` (web/app/components/workflow/header/view-history.tsx line 69)
  - **方法**: GET

8. singleNodeRun

- **用途**: 运行单个节点。
- **属性**:
  - `appId: string` - 应用 ID。
  - `nodeId: string` - 节点 ID。
  - `params: object` - 运行参数。
- **返回值**: `Promise<any>`
  - **用例：** `web/app/components/workflow/nodes/_base/hooks/use-one-step-run.ts line 199`
- **后端交互**:
  - **URL**: `/apps/${appId}/workflows/draft/nodes/${nodeId}/run`
  - **方法**: POST

9. getIterationSingleNodeRunUrl

- **用途**: 获取单个节点运行的 URL。
- **属性**:
  - `isChatFlow: boolean` - 是否为聊天流。
  - `appId: string` - 应用 ID。
  - `nodeId: string` - 节点 ID。
- **返回值**: `string` - URL 字符串

10. stopWorkflowRun

- **用途**: 停止工作流运行。
- **属性**:
  - `url: string` - 请求的 URL。
- **返回值**: `Promise<CommonResponse>`
  - **`CommonResponse`**
    - `result: 'success' | 'fail'` - 请求的结果。
- **后端交互**:
  - **URL**: {URL}
    - **用例：** `/apps/${appId}/workflow-runs/tasks/${taskId}/stop` (web/app/components/workflow/hooks/use-workflow-run.ts line 479)
  - **方法**: POST

11. fetchNodeDefault

- **用途**: 获取节点的默认配置。
- **属性**:
  - `appId: string` - 应用 ID。
  - `blockType: BlockEnum` - 节点类型。
  - `query: object` - 查询参数。
- **返回值**: `Promise<any>`
- **后端交互**:
  - **URL**: `/apps/${appId}/workflows/default-workflow-block-configs/${blockType}`
  - **方法**: GET