**技术方案v0.1**

相关文档：

产品需求文档

名词解释：

* 用户：系统中的任意账户
* 客户：特指非测试中心成员、作为测试委托发起方的一类用户

**1 关键问题**

**1.0 技术约定**

**基本约定**

* 使用layout配置：[**@alipay/umi-plugin-layout**](https://umijs.org/zh-CN/plugins/plugin-layout)
* 数据请求使用useRequest：[网络请求 - Ant Design Pro](https://pro.ant.design/zh-CN/docs/request)
* 使用框架内置的JSlint
* npm run lint
* 使用less作为样式语言，注意框架默认使用了[CSS Modules](https://pro.ant.design/zh-CN/docs/css-modules)
* less中文文档：[Less 快速入门 | Less.js 中文文档 - Less 中文网](https://less.bootcss.com/) 基本和css一样，不用多学什么

**接口规范**

本系统中统一使用[@umijs/plugin-request](https://umijs.org/zh-CN/plugins/plugin-request#userequest)进行数据请求；

**后端接口路径统一添加 /api 的前缀。接口数据格式统一为：**

|  |
| --- |
| JSON  {  "success": true,  "data": {},  "errorMessage": "error message",  //以下不必须：  "errorCode": "1001",  "showType": 2,  "traceId": "someid",  "host": "10.1.1.1"  } |

* @苏晨阳 和后端组对接

**1.1 角色、功能与菜单**

**技术说明**

* 本系统需采取RBAC模式进行权限管理。首先分别建立角色树与功能树，建立：
* 用户到角色树叶节点的多对多映射；
* 角色树叶节点到功能树子树的多对多映射；

**本系统中，将具体功能树与菜单进行统一，所以角色树对应的功能树子树必须是从根节点开始的。**

用户登录时，后端查询RBAC获取对应功能树并生成对应菜单，以JSON格式发送给前端。也即，**前端的菜单是从服务端请求获得。**

框架中提供了生成菜单的API：menu.request 和 menu.params，request 需要传入一个 promise，它会自动托管 loading，params 修改会触发 request 方便重新请求菜单。

在 src/app.tsx 定义 layout 对象，并且导出。其中initialState 包括了角色信息，是本系统的[全局初始化数据](https://pro.ant.design/zh-CN/docs/initial-state)，参考 [@umijs/plugin-initial-state](https://umijs.org/zh-CN/plugins/plugin-initial-state)。示例：

|  |
| --- |
| TypeScript  // https://umijs.org/zh-CN/plugins/plugin-layout  export const layout: RunTimeLayoutConfig = ({ initialState }) => {  return {  menu: {  // 每当 initialState?.currentUser?.userid 发生修改时重新执行 request  params: {  userId: initialState?.currentUser?.userid,  },  request: async (params, defaultMenuData) => {  // initialState.currentUser 中包含了所有用户信息  const menuData = await fetchMenuData();  return menuData;  },  },  };  }; |

**初步的角色树**

* 超级管理员
* 工作人员
* 市场部
* 中级人员
* 初级人员
* ...
* 测试部
* 质量部
* ...
* 客户

**初步的功能树**

* 系统管理
* RBAC系统管理（“用户管理”）
* 系统一般性配置（菜单、样式、邮件系统等）
* 客户管理
* 系统中客户的CRUD
* 委托管理
* 系统中委托的CRUD
* ...
* 待补充

**1.2 前端的权限管理**

参考：https://pro.ant.design/zh-CN/docs/authority-management

1.1维护了针对具体用户的功能树数据结构，并据此生成对应菜单，但前端并未真正实现权限管理。用户对不在菜单中的页面的访问仍未受限制，且需解决不同角色用户在同一页面中可以看到的元素和进行的操作不同这一问题。前端的页面管理包括：

* 页面内权限管理
* 菜单和路由的权限管理

此处使用插件：

* [@umijs/plugin-access](https://umijs.org/zh-CN/plugins/plugin-access)
* [@alipay/umi-plugin-layout](https://umijs.org/zh-CN/plugins/plugin-layout)

创建 src/access.ts ，在该文件中 export default 一个函数，定义用户拥有的权限，示例：

|  |
| --- |
| TypeScript  // src/access.ts  export default function (initialState) {  return {  canReadFoo: true,  canUpdateFoo: () => true,  canDeleteFoo: (data) => data?.status < 1, // 按业务需求自己任意定义鉴权函数  };  } |

该文件需要返回一个 function，返回的 function 会在应用初始化阶段被执行，执行后返回的对象将会被作为用户所有权限的定义。对象的每个 key 对应一个 boolean 值，只有 true 和 false，代表用户是否有该权限。

页面内的权限管理示例：

|  |
| --- |
| TypeScript  import React from 'react';  import { useAccess, Access } from 'umi';    const PageA = (props) => {  const { foo } = props;  const access = useAccess(); // access 实例的成员: canReadFoo, canUpdateFoo, canDeleteFoo    if (access.canReadFoo) {  // 任意操作  }    return (  <div>  <Access accessible={access.canReadFoo} fallback={<div>Can not read foo content.</div>}>  Foo content.  </Access>  <Access accessible={access.canUpdateFoo()} fallback={<div>Can not update foo.</div>}>  Update foo.  </Access>  <Access accessible={access.canDeleteFoo(foo)} fallback={<div>Can not delete foo.</div>}>  Delete foo.  </Access>  </div>  );  }; |

菜单和路由的权限管理可以参考：https://pro.ant.design/zh-CN/docs/authority-management，注意需遵循框架的全局初始数据和路由配置。

**2 文件结构**

待功能树完善后补充

**3 全局组件**

包括需要在框架层面上进行重构的部分