NewISpider系统前后端接口约定

> version: 0.0.1

本文档描述的是 NewISpider系统 的**前端页面和后端接口**之间的通讯方式. 后端各个模块之间的调用,不在本文档描述范围内.

一般性约定

一般性约定作为系统的默认约定,是在相关接口文档未进行说明时的惯例和标准. 在和接口文档的本身的约定冲突时,以接口文档本身的定义为准

- 字符集UTF-8
- 接口至少支持POST方法
- CORS(Cross-Origin Resource Sharing) 跨域问题由后端处理
- 跳转问题由前端处理
- 请求体和返回体中使用1/0(整形)替代True/False(布尔类型)
- 请求头参数名小写,如需要使用连字符号,请使用"-"而不是"_"
- 身份信息使用请求头(request.headers)中的authorization(小写)参数验证.(详细验证方法见后继文档)
- 开发环境可以使用http协议.生产环境使用https协议
- 返回体格式JSON.
    1. 返回{"message": "success"}表示请求顺利完成并返回了正确的信息.
    2. 返回{"message": "because something is error....."} 请求行为顺利完成,但返回了错误的信息
    3. 返回{"message": "401"} 表示本次请求需要提供身份验证信息(原始的身份信息可能已过期,注意401是字符串格式)
    4. 返回{"message": "rejected"} 表示当前用户权限不足,操作被拒绝.
    5. 具体示范见*返回体示例*

示例

以下示例仅为了展示处理的逻辑,并非唯一或者最佳行为指南.

预定义函数

预定义3个函数,用于简化后面的处理过程:

- **get_auth**: 从会话存储/本地存储中取出authorization信息.
- **save_auth**: 将authorization信息写入会话存储/本地存储.
- **post_plus**: 一个自定义的post函数,可以在执行post操作的时候自定义请求头中的authorization字段

```
/*
将authorization信息写入会话存储
*/

var save_auth = function(authorization){
    sessionStorage.setItem("authorization", authorization);
};

/*
将authorization信息从会话存储取出
*/

var get_auth = function(){
    return sessionStorage.getItem("authorization");
};

/*
自定义一个可以添加请求头的post函数.
*/
var post_plus = function(url, args, success_callback, error_callback, authorization){
    /*
    url: 请求地址
    args: 请求参数字典
    success_callback: 成功的回调函数
    error_callback: 失败的回调函数
    authorization: 身份验证信息(jwt的密文)
    */

    // 取 authorization 的值
    var auth = authorization === undefined? sessionStorage.getItem("authorization");

    // 自定义post请求
    var setting = {
        url: url,
        type: "POST",
        data: args,
        headers: {"authorization": auth},
        success: success_callback,
        error: error_callback
    };

    $.ajax(setting);

};
```

登录并保存authorization

```
var url = "/manage/login";
// 登录接口不需要authorization信息
var args = {
    "user_name": "user_01",
    "password": $.md5("123456")    // 不能传送明文密码.$.md5时一个jQuery的md5扩展.
};

/*
url: 请求的地址
args: 请求的参数
response: 原始的返回体
*/
$.post(url, args, function(response){
    var json = JSON.parse(response);
    var mes = json['message'];
    if(mes === "success"){
        // 成功.
        var auth = mes['authorization'];
        save_auth(auth);  // 保存authorization信息
    }
    else if(mes === "401"){
        // 需要登录
        location.href = "/login";
    }
    else if(mes === "rejected"){
        // 权限不足,操作被拒绝
        ....
    }
    else{
        // 出错了.
        alert(mes);    // 弹出错误信息
        return false;
    }
});
```

前端处理请求返回体示例

```
var url = "/manage/user_info";
var args = {
    "user_id": 12
};

/*
url: 请求的地址
args: 请求的参数
response: 原始的返回体.json格式,一般是:
    请求成功:  {'message': 'success', 'authorization': 'eyJ0eXAil...'}
    请求失败:  {'message': 'error', 'authorization': 'eyJ0eXAil...'}
    authorization是下一次请求时用到的有效的身份信息.
*/
post_plus(url, args, function(response){
    /*
    post_plus会自动取出sessionStorage中保存的authorization信息.
    后端会以请求头中authorization信息的来确认请求着的身份和权限.
    */
    var json = JSON.parse(response);
    var mes = json['message'];
    if(mes === "success"){
        // 成功.
        save_auth(json['authorization']); // 更新authorization信息
        ....                              // 执行其它代码
    }
    else if(mes === "401"){
        // 需要登录
        location.href = "/login";
    }
    else if(mes === "rejected"){
        // 权限不足,操作被拒绝
        ....
    }
    else{
        // 出错了.
        alert(mes);    // 弹出错误信息
        return false;
    }
});
```

后端处理前端请求的逻辑

```
"""
为了简便,以flask做个示范.
"""
from flask import Flask
from flask import request
import json
app = Flask(__name__)


@app.route("/customer_info", methods=['post', 'get'])
@check_auth    # 检查身份的装饰器,返回当前用户信息和对应权限的值
def func(user: dict, permission: int, next_auth: str):
    """
    查询顾客信息
    param user:       当前操作者的信息
    param permission: 整形数据, 0/1 代表是否有此权限
    param next_auth:  新分配的authorization
    return: json      格式数据
    """
    resp = {"message": "success", "authorization": authorization}
    if permission == 1:
        # 允许操作
        customer_id = request.form.get("customer_id")  # 从请求中取出customer_id
        data = get_customer_info(customer_id=customer_id)   # 查询顾客信息
        resp['data'] = data
    else:
        resp['message'] = "rejected"   # 权限不足
    return json.dumps(resp)            # 返回json
```

分页查询的请求和返回

请求体

分页查询有2个通用的参数

参数名	数据类型	含义	说明
page	int	当前页面	非必须,默认第一页
page_size	int	每页多少条记录	非必须,会提供默认值

返回体

返回体的格式为{"message": "success", "data": data} 其中data时分页查询的结果的字典

```
// 请求的示范,查询第三页数据,每页最多15条记录
var args = {
    page: 3,
    page_size: 15,
}
$.get(url, args, function(resp){
    // your code.......
});

// 分页返回的示范
$.post(url, args, function(resp){
    var json = JSON.parse(resp);
    var status = json['message'];
    if(status === "success"){
        var data = json['data'];
        var count = data['count'];       // 共计多少条记录
        var prev_index = data['previous'];  // 上一页url,可能为null
        var next_index = data['next'];  // 下一页url,可能为null
        var results = data['results'];   // 数据集
    }
});
```