API 接口开放平台

项目地址

背景：

1. 前端开发需要用到后端接口
2. 要用现成的接口

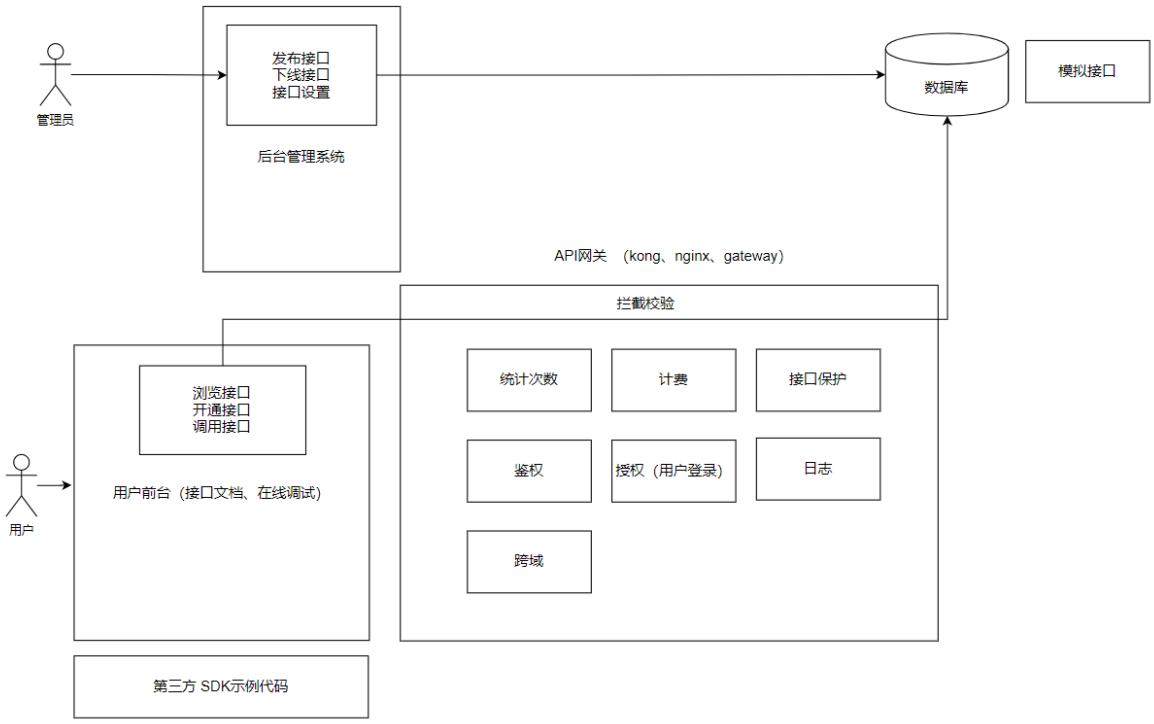
做一个API接口平台：

1. 防止攻击（安全性）
2. 不能随便调用（限制、开通）
3. 调用次数
4. 计费
5. 流量保护
6. API接入

项目介绍

做一个提供API接口调用的平台，用户可以注册登录，开通接口调用权限。用户可以使用接口，并且每次调用会进行统计。管理员可以发布接口、下线接口、接入接口，以及可视化接口的调用情况、数据。

> x项目流程图



技术选型

前端

Ant Design Pro

React

Ant Design Procomponents

Umi

Umi Requset (Axios 的封装)

后端

java Spring boot

Spring Boot Starter (SDK 开发)

？？？（网关、限流、日志实现）

项目分期

初始化和展示(第一部分)

项目介绍、设计、技术选型

基础项目的搭建

接口管理

用户查看接口

接口调用(第二部分)

1. 开发模拟API接口
2. 开发调用这个接口的代码
3. 保证调用的安全性（API签名认证）
4. 客户端 SDK 的开发
5. 管理员接口 **发布** 与调用
6. 接口文档展示、接口在线调用

接口计费与保护(第三部分)

统计用户调用次数

限流

计费

日志

开通

管理、统计分析(第四部分)

提供可视化平台，用图表的方式展示所有的接口的调用情况，便于调整业务。

第一部分

主要内容：antd 前端框架的整合，踩坑，openApi的使用

Antd Pro 框架操作 & 踩坑

框架安装

```
npm i @ant-design/pro-cli -g
```

项目搭建

```
pro create 项目名称
```

移除国际化报错

> 错误一

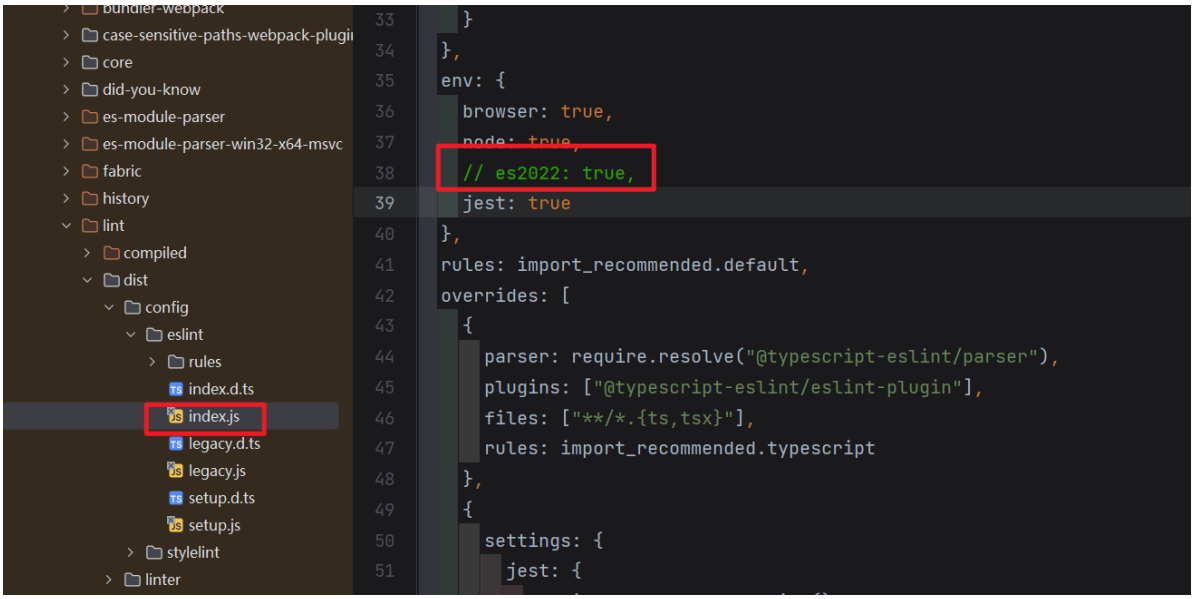
```
// 报错 prettier
npm i eslint prettier-eslint eslint-config-prettier --save-dev
```

> 错误二

```
Error: .eslintrc.js » E:\code\yupi-java\Api-interface\code\
        Environment key "es2022" is unknown
```

解决办法

[解决博客]




```
npm i @typescript-eslint/eslint-plugin
npm i eslint-plugin-eslint-comments
npm i eslint-plugin-jest
npm i eslint-plugin-unicorn
```

成功解决

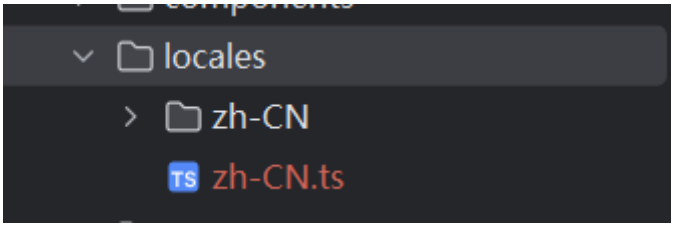
解决后将原本的i18n目录删除 注意 看下面保存一下这个文件

国际化遗留问题

到最后还是出现了问题，索性国际化就没再移除—

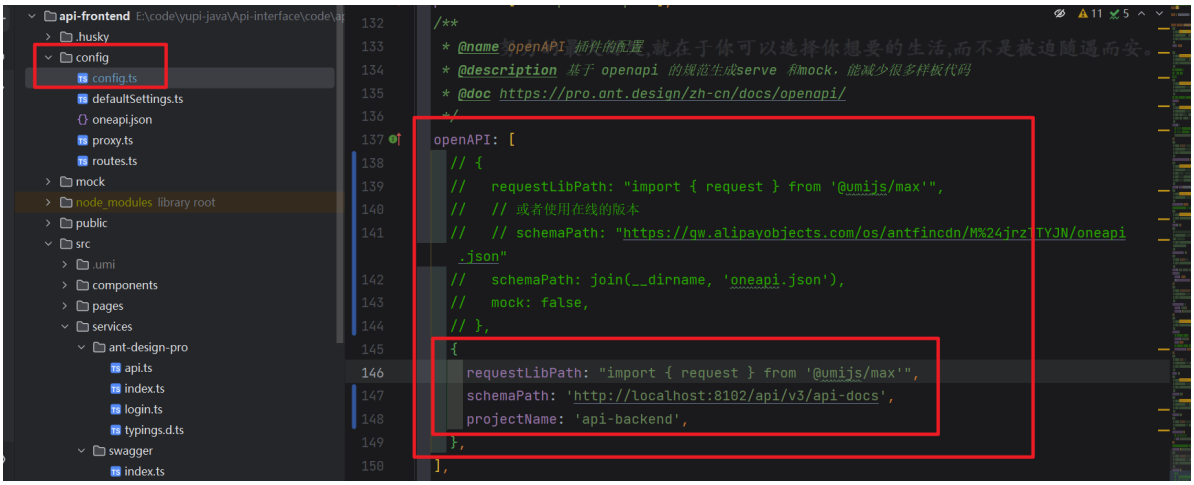
```
    at Browser (http://localhost:8000/mf-dep___vendor.c2bb1d9f.js:111692:68)
▼[React Intl] Missing message: "pages.login.forgotPassword" for locale: "zh-CN", using default message as fallback.   index.js:1
    at FormattedMessage (http://localhost:8000/mf-dep___vendor.c2bb1d9f.js:108238:181)
    at a
```

可以将这个汉语的文件添加回来



Antd 快速与后端增删改查对应

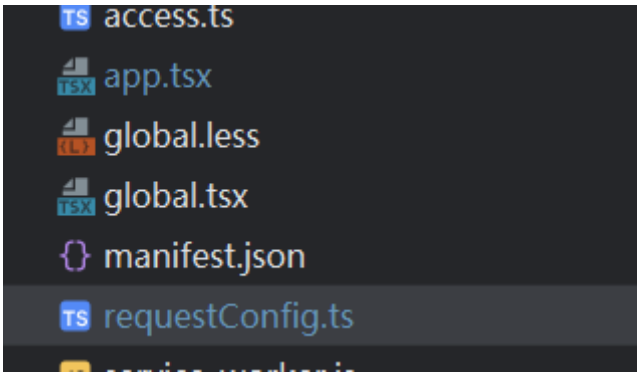
对应后端的 openApi 的地址接口，自动生成



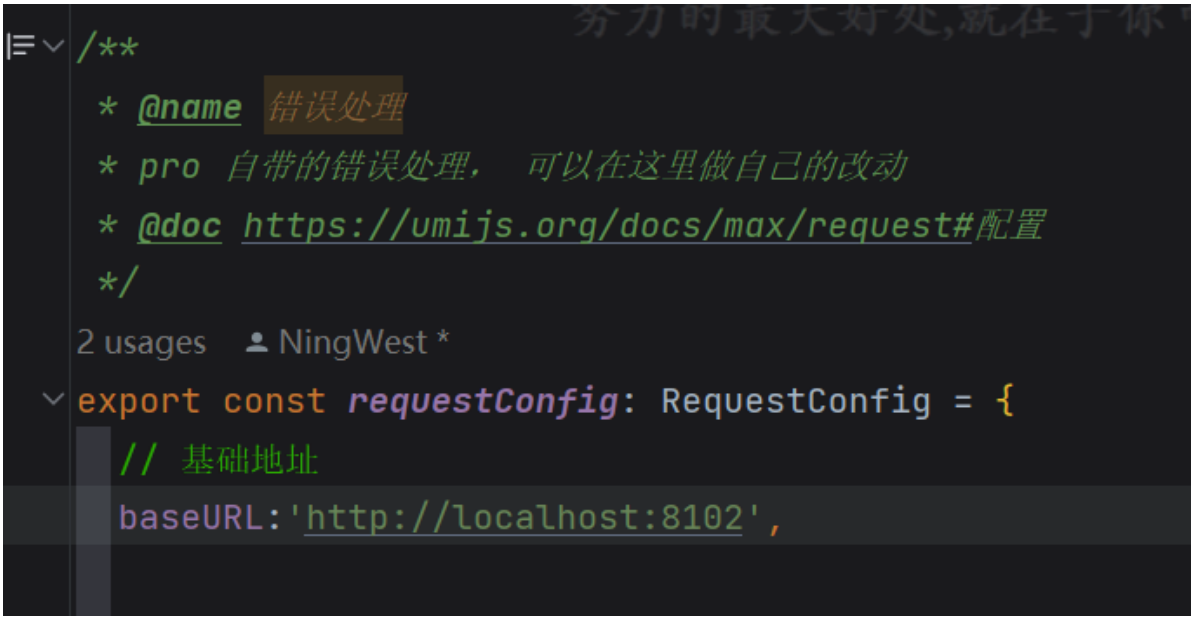
> 点击即可生成代码

注意：后端的controller上不要有 @Api(tags = "微信公众号相关接口") 注解 ，前端生成代码的文件名会显示这里面汉字的拼音。。。

Antd 修改请求的配置



将原本的 requestErrorConfig 修改为 requestConfig



在requestConfig中添加 基础地址

修改Antd框架的登录接口

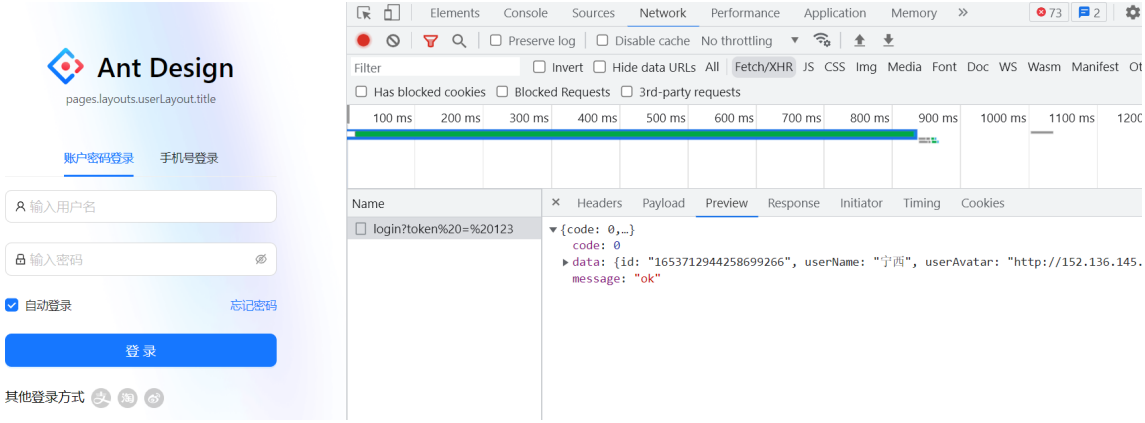
> 登录逻辑修改

```
const handleSubmit = async (values: API.UserLoginRequest) => {
    try {
      // 登录  修改为自己后端的登录接口
      const res = await userLoginUsingPOST({ ...values });
      if (res.data) {
        const urlParams = new URL(window.location.href).searchParams;
        history.push(urlParams.get('redirect') || '/');
        return;
      }
      // 如果失败去设置用户错误信息
      setUserLoginState(msg);
```

```
    } catch (error) {
      const defaultLoginFailureMessage = intl.formatMessage({
        id: 'pages.login.failure',
        defaultMessage: '登录失败，请重试！',
      });
      console.log(error);
      message.error(defaultLoginFailureMessage);
    }
  };
```

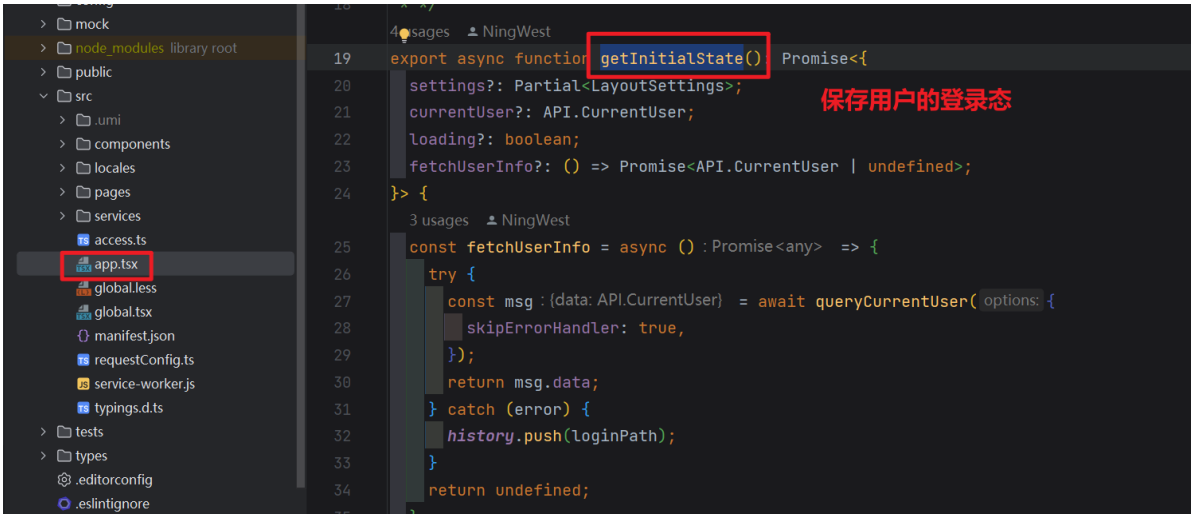
注意：将项目中的所有 的登录对应的参数都要换成自己的，例如登录参数的接受实体信息，antd框架的登录信息是 username，password，我们自己的是userAccount，userPassword，都要修改。

> 修改完毕之后登录即可使用，但是因为没有保存用户的登录态，还是无法进入信息页面

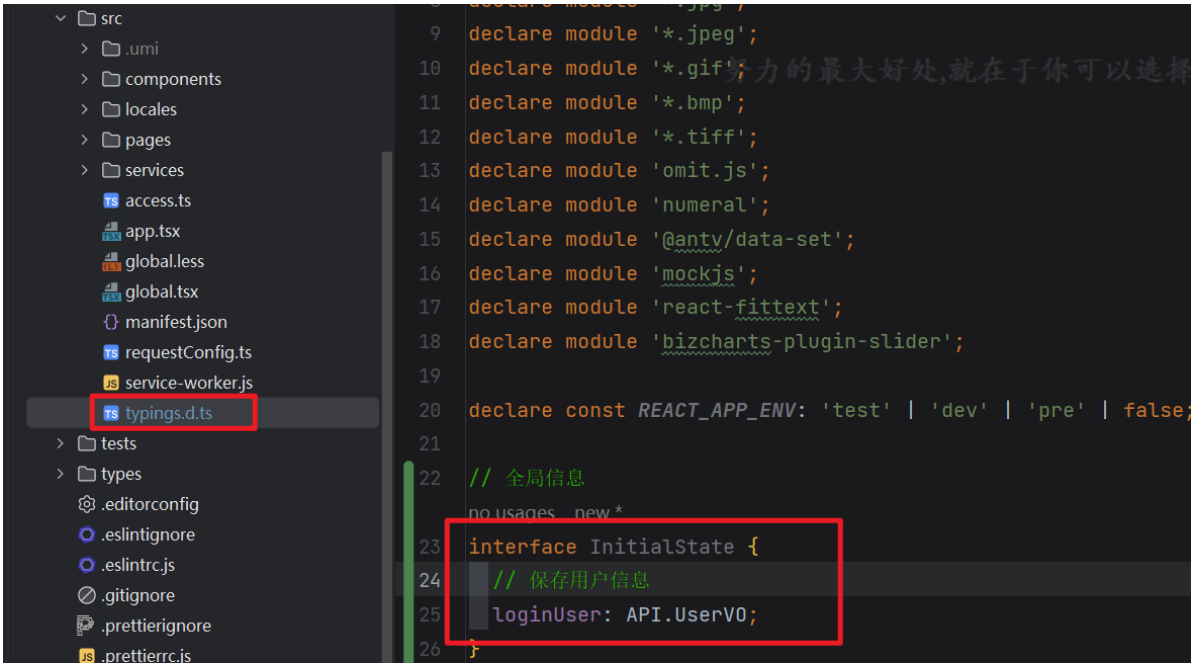


Antd 保存用户的登录态

> 保存信息的方法



> 保存信息的类型



> 登录完毕之后保存用户的登录信息

```
const handleSubmit = async (values: API.UserLoginRequest) => {
  try {
    // 登录  修改为自己后端的登录接口
    const res = await userLoginUsingPOST({ ...values });
    if (res.data) {
      const urlParams = new URL(window.location.href).searchParams;
      history.push(urlParams.get('redirect') || '/');
      // 登录  ------------------
      setInitialState({
        loginUser: res.data,
      });
      // ----------------------
      return;
    }
  } catch (error) {
    const defaultLoginFailureMessage = intl.formatMessage({
      id: 'pages.login.failure',
      defaultMessage: '登录失败，请重试！',
    });
    console.log(error);
    message.error(defaultLoginFailureMessage);
  }
};
```

修改完毕之后，在登录即可跳转页面‘

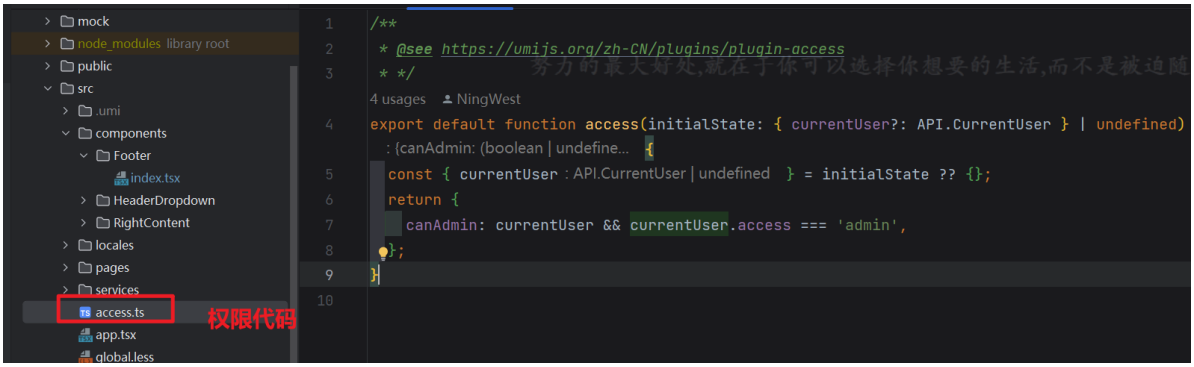
> 刷新页面，跳转登录页的问题，修改这个方法

```
};
// 如果不是登录页面，执行  问题解决
const {location : Location } = history;
if (location.pathname !== loginPath) {
  const res : API.BaseResponseLoginUserVO  = await getLoginUserUsingGET();
  if (res.data) {
    statue.loginUser = res.data
  }
  // 返回信息
  return statue;
}
```

> 这两个方法，这样写可行

```
1 usage
const fetchUserInfo = async () : Promise<InitialState>  => {
  try {
    const res : API.BaseResponseLoginUserVO  = await getLoginUserUsingGET();
    if (res.data) {
      statue.loginUser = res.data
    }
  } catch (error) {
    history.push(loginPath);
  }
  // 返回信息
  return statue;
};
// 如果不是登录页面，执行  问题解决
const {location : Location } = history;
if (location.pathname !== loginPath) {
  const res : API.BaseResponseLoginUserVO  = await getLoginUserUsingGET();
  if (res.data) {
    statue.loginUser = res.data
  }
  // 返回信息
  return statue;
}
```

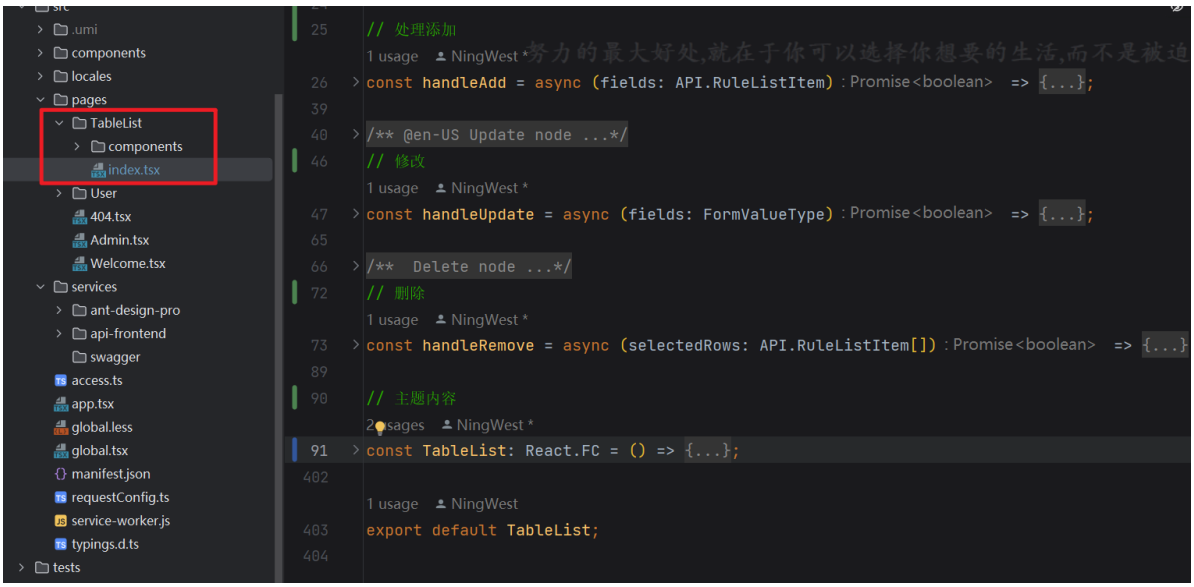
Antd 处理角色问题



> 修改代码

```
export default function access(initialState: InitialState | undefined) {
  const {loginUser} = initialState ?? {};

  return {
    canAdmin: loginUser && loginUser.userRole === 'admin',
    canUser: loginUser && loginUser.userRole === 'user',
  };
}
```

Antd 表格使用



> 修改内容

```
import {addRule, removeRule, updateRule} from '@/services/ant-design-pro/api';
import {PlusOutlined} from '@ant-design/icons';
import type {ActionType, ProColumns, ProDescriptionsItemProps} from '@ant-design/pro-components';
import {
```

```
  FooterToolbar,
  ModalForm,
  PageContainer,
  ProDescriptions,
  ProFormText,
  ProFormTextArea,
  ProTable,
} from '@ant-design/pro-components';
import {FormattedMessage, useIntl} from '@umijs/max';
import {Button, Drawer, message} from 'antd';
import React, {useRef, useState} from 'react';
import type {FormValueType} from './components/UpdateForm';
import UpdateForm from './components/UpdateForm';
import {listInterfaceInfoVOByPageUsingGET} from "@/services/api-
frontend/interFaceInfoController";

/**
 * @en-US Add node
 * @zh-CN 添加节点
 * @param fields
 */

// 处理添加
const handleAdd = async (fields: API.RuleListItem) => {
  const hide = message.loading('正在添加');
  try {
    await addRule({...fields});
    hide();
    message.success('Added successfully');
    return true;
  } catch (error) {
    hide();
    message.error('Adding failed, please try again!');
    return false;
  }
};

/**
 * @en-US Update node
 * @zh-CN 更新节点
 *
 * @param fields
 */
// 修改
const handleUpdate = async (fields: FormValueType) => {
  const hide = message.loading('Configuring');
  try {
    await updateRule({
      name: fields.name,
      desc: fields.desc,
      key: fields.key,
    });
    hide();

    message.success('Configuration is successful');
```

```
    return true;
  } catch (error) {
    hide();
    message.error('Configuration failed, please try again!');
    return false;
  }
};

/**
 *  Delete node
 * @zh-CN 删除节点
 *
 * @param selectedRows
 */
// 删除
const handleRemove = async (selectedRows: API.RuleListItem[]) => {
  const hide = message.loading('正在删除');
  if (!selectedRows) return true;
  try {
    await removeRule({
      key: selectedRows.map((row) => row.key),
    });
    hide();
    message.success('Deleted successfully and will refresh soon');
    return true;
  } catch (error) {
    hide();
    message.error('Delete failed, please try again');
    return false;
  }
};

// 主题内容
const TableList: React.FC = () => {
  /**
   * @en-US Pop-up window of new window
   * @zh-CN 新建窗口的弹窗
   *  */
  const [createModalOpen, handleModalOpen] = useState<boolean>(false);
  /**
   * @en-US The pop-up window of the distribution update window
   * @zh-CN 分布更新窗口的弹窗
   * */
  const [updateModalOpen, handleUpdateModalOpen] = useState<boolean>
(false);

  const [showDetail, setShowDetail] = useState<boolean>(false);

  const actionRef = useRef<ActionType>();
  const [currentRow, setCurrentRow] = useState<API.RuleListItem>();
  const [selectedRowsState, setSelectedRows] = useState<API.RuleListItem[]>
([]);

  /**
   * @en-US International configuration
```

```
   * @zh-CN 国际化配置
   * */
  const intl = useIntl();

  const columns: ProColumns<API.InterfaceInfoVo>[] = [
    {
      title: "id",
      dataIndex: 'id',
      valueType: 'index',
    },
    {
      title: "接口名",
      dataIndex: 'name',
      valueType: 'text'
    },
    {
      title: "url",
      dataIndex: 'url',
      valueType: 'textarea',
    },
    {
      title: "请求类型",
      dataIndex: 'method',
      valueType: 'textarea',
    },
    {
      title: "创建人",
      dataIndex: 'userName',
      valueType: 'text'
    },
    {
      title: "响应头",
      dataIndex: 'responseHeader',
      valueType: 'textarea',
    },
    {
      title: "接口描述",
      dataIndex: 'description',
      valueType: 'textarea',
    },
    {
      title: "请求头",
      dataIndex: 'requestHeader',
      valueType: 'textarea',
    },
    {
      title: "接口状态",
      dataIndex: 'status',
      hideInForm: true,
      valueEnum: {
        0: {
          text: "关闭",
          status: 'Default',
        },
        1: {
```

```
          text: "开启",
          status: 'Processing',
        }
      },
    },
    {
      title: "创建时间",
      dataIndex: 'createTime',
      valueType: 'dateTime',
    },
    {
      title: "操作",
      dataIndex: 'option',
      valueType: 'option',
      render: (_, record) => [
        <a
          key="config"
          onClick={() => {
            handleUpdateModalOpen(true);
            setCurrentRow(record);
          }}
        >
          <FormattedMessage id="pages.searchTable.config"
defaultMessage="Configuration"/>
        </a>,
        <a key="subscribeAlert" href="https://procomponents.ant.design/">
          <FormattedMessage
            id="pages.searchTable.subscribeAlert"
            defaultMessage="Subscribe to alerts"
          />
        </a>,
      ],
    },
  ];

  return (
    <PageContainer>
      <ProTable<API.RuleListItem, API.PageParams>
        headerTitle={intl.formatMessage({
          id: 'pages.searchTable.title',
          defaultMessage: 'Enquiry form',
        })}
        actionRef={actionRef}
        // key报错
        rowKey="id"
        search={{
          labelWidth: 120,
        }}
        toolBarRender={() => [
          <Button
            type="primary"
            key="primary"
            onClick={() => {
              handleModalOpen(true);
            }}
```

```
          >
            <PlusOutlined/> <FormattedMessage id="pages.searchTable.new" defaultMessage="New"/>
          </Button>,
        ]}
        // 更改自己获取数据的方法
        // @ts-ignore
        request={async (params) => {
          let res = await listInterfaceInfoVOByPageUsingGET({
            ...params
          })
          if (res.data) {
            return {
              data: res.data?.reconds || [],
              success: true,
              total: res.data.total,
            }
          }
        }}
        columns={columns}
        rowSelection={{
          onChange: (_, selectedRows) => {
            setSelectedRows(selectedRows);
          },
        }}
      />
      {selectedRowsState?.length > 0 && (
        <FooterToolbar
          extra={
            <div>
              <FormattedMessage id="pages.searchTable.chosen" defaultMessage="Chosen"/>{' '}
              <a style={{fontWeight: 600}}>{selectedRowsState.length}</a>{' '}
              <FormattedMessage id="pages.searchTable.item" defaultMessage="项"/>
                
              <span>
                <FormattedMessage
                  id="pages.searchTable.totalServiceCalls"
                  defaultMessage="Total number of service calls"
                />{' '}
                {selectedRowsState.reduce((pre, item) => pre + item.callNo!, 0)}{' '}
                <FormattedMessage id="pages.searchTable.tenThousand" defaultMessage="万"/>
              </span>
            </div>
          }
        >
          <Button
            onClick={async () => {
              await handleRemove(selectedRowsState);
              setSelectedRows([]);
              actionRef.current?.reloadAndRest?.();
```

```
            }}
          >
            <FormattedMessage
              id="pages.searchTable.batchDeletion"
              defaultMessage="Batch deletion"
            />
          </Button>
          <Button type="primary">
            <FormattedMessage
              id="pages.searchTable.batchApproval"
              defaultMessage="Batch approval"
            />
          </Button>
        </FooterToolbar>
      )}
      <ModalForm
        title={intl.formatMessage({
          id: 'pages.searchTable.createForm.newRule',
          defaultMessage: 'New rule',
        })}
        width="400px"
        open={createModalOpen}
        onOpenChange={handleModalOpen}
        onFinish={async (value) => {
          const success = await handleAdd(value as API.RuleListItem);
          if (success) {
            handleModalOpen(false);
            if (actionRef.current) {
              actionRef.current.reload();
            }
          }
        }}
      >
        <ProFormText
          rules={[
            {
              required: true,
              message: (
                <FormattedMessage
                  id="pages.searchTable.ruleName"
                  defaultMessage="Rule name is required"
                />
              ),
            },
          ]}
          width="md"
          name="name"
        />
        <ProFormTextArea width="md" name="desc" />
      </ModalForm>
      <UpdateForm
        onSubmit={async (value) => {
          const success = await handleUpdate(value);
          if (success) {
            handleUpdateModalOpen(false);
```

```
            setCurrentRow(undefined);
            if (actionRef.current) {
              actionRef.current.reload();
            }
          }
        }}
        onCancel={() => {
          handleUpdateModalOpen(false);
          if (!showDetail) {
            setCurrentRow(undefined);
          }
        }}
        updateModalOpen={updateModalOpen}
        values={currentRow || {}}
      />

      <Drawer
        width={600}
        open={showDetail}
        onClose={() => {
          setCurrentRow(undefined);
          setShowDetail(false);
        }}
        closable={false}
      >
        {currentRow?.name && (
          <ProDescriptions<API.RuleListItem>
            column={2}
            title={currentRow?.name}
            request={async () => ({
              data: currentRow || {},
            })}
            params={{
              id: currentRow?.name,
            }}
            columns={columns as ProDescriptionsItemProps<API.RuleListItem>
[]}
          />
        )}
      </Drawer>
    </PageContainer>
  );
};

export default TableList;

```

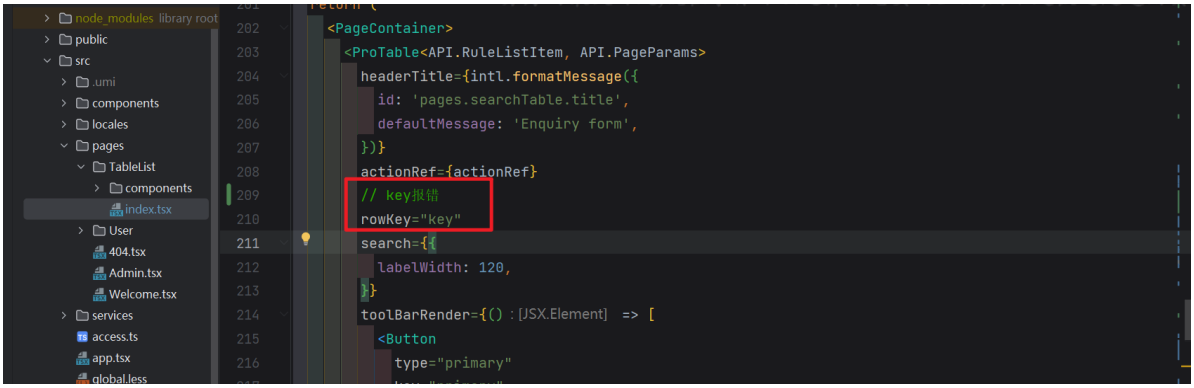
> 获取数据

```
request={rule}
```

请求数据的接口

key报错问题

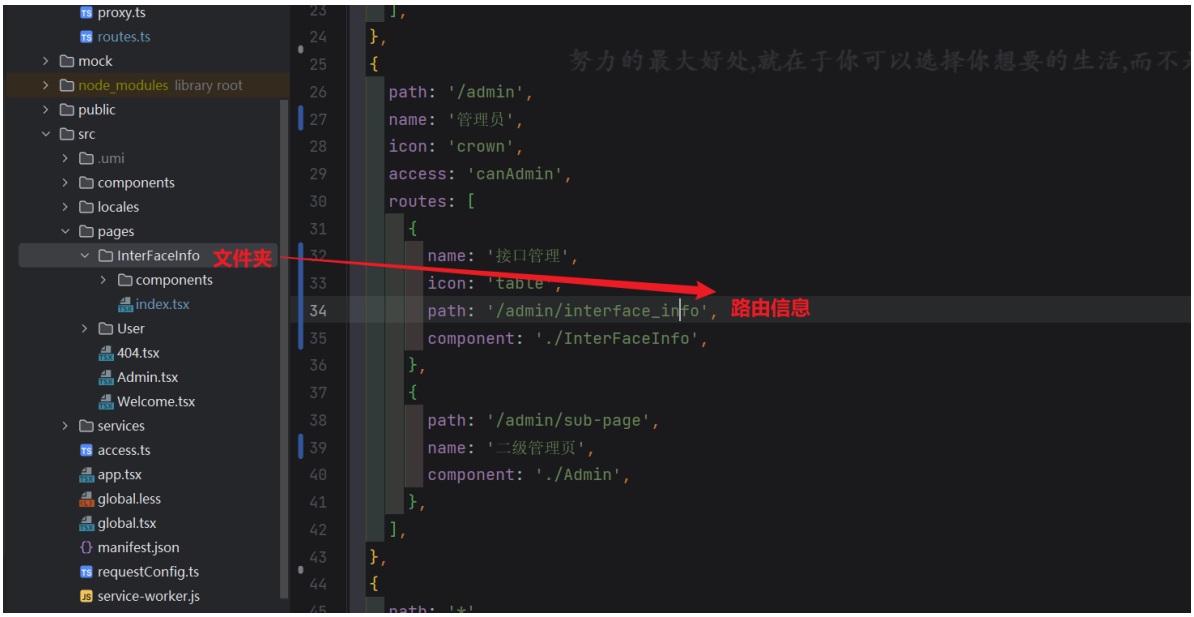
```
▶ Warning: Each child in a list should have a unique "key" prop.                    index.j

Check the render method of `Body`. See https://reactjs.org/link/warning-keys for more information.
    at ImmutableComponent (http://localhost:8000/mf-dep____vendor.f28edef7.js:105156:5)
    at Body (http://localhost:8000/mf-dep____vendor.f28edef7.js:240165:70)
    at ImmutableComponent (http://localhost:8000/mf-dep____vendor.f28edef7.js:105156:5)
    at table
    at div
    at div
    at div
    at Provider (http://localhost:8000/mf-dep____vendor.f28edef7.js:105198:22)
```



> 解决办法

```
1  rowKey="id"
```

路由修改



对应组件的名称

Antd模态框

> 封装组件

```
 <CreateForm
    open={createModalOpen}
    columns={columns}
    // 关闭
    onCancel={() => {
      // 关闭模态框
      handleModalOpen(false)
    }}
    onSubmit={async (value) => {
      const success = await handleAdd(value);
      if (success) {
        handleUpdateModalOpen(false);
        setCurrentRow(undefined);
        if (actionRef.current) {
          actionRef.current.reload();
        }
      }
    }}
  />
```

> 组件内部

```
import {ProColumns, ProTable,} from '@ant-design/pro-components';
import React from 'react';
import {Modal} from "antd";

export type Props = {
  // 将列全部传递过来
  columns: ProColumns<API.InterfaceInfoVo>[],
  onCancel: () => void;
  // @ts-ignore
  onSubmit: (values: API.InterFaceInfo) => Promise<void>;
  open: boolean;
};

const CreateFrom: React.FC<Props> = (props) => {
  const {open, onCancel, onSubmit, columns} = props
  return (
    <Modal open={open} onCancel={() => onCancel?.()}>
      <ProTable
        type="form"
        columns={columns}
        onSubmit={async (value) => {
          onSubmit?.(value)
        }}
      />
    </Modal>
```

```
  )
};

export default CreateFrom;
```

Antd不展示某行

> hideInForm 不展示某行

```
{
  title: "创建时间",
  dataIndex: 'createTime',
  valueType: 'dateTime',
  hideInForm: true
},
```

Antd设置规则

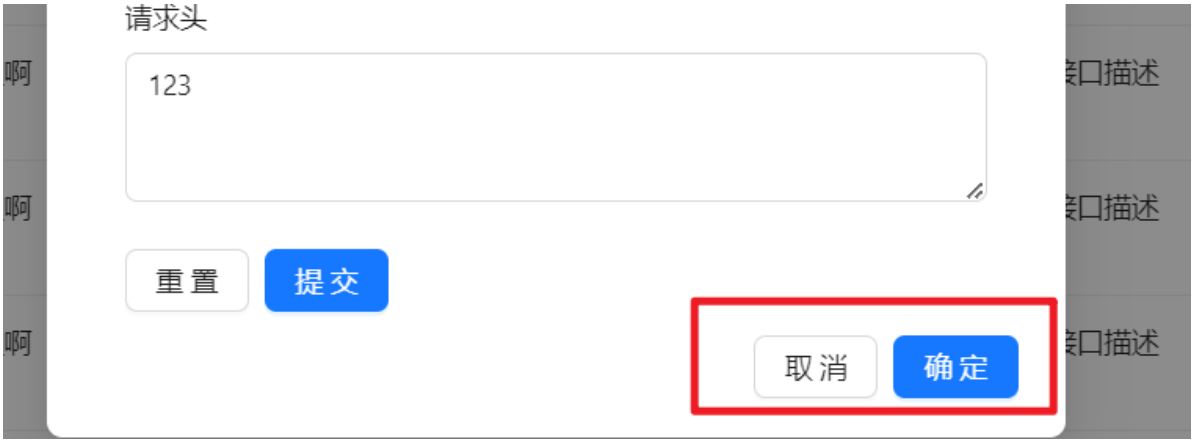
formItemProps

```
{
  title: "接口名",
  dataIndex: 'name',
  valueType: 'text',
  formItemProps: {
    rules: [{
      required: true,
      message: '接口名必须填写！'
    }]
  }
},
```

Antd 设置表格

> footer={null}

```
<Modal footer={null} open={open} onCancel={() => onCancel?.()}>
  <ProTable
    type="form"
    columns={columns}
    onSubmit={async (value) => {
      onSubmit?.(value)
    }}
  />
</Modal>
```



修改接口无法获取id问题?

> 问题描述?

在下面的方法中，在这个fields 中无法获取 id？**我看了又看，两眼一抹黑**，自己想了个办法 从 currentRow 中获取 id，

```
const handleUpdate = async (fields: API.InterfaceInfoVo) => {
    const hide = message.loading('修改中。');
    try {
      // @ts-ignore
      await updateInterfaceInfoUsingPOST({...fields, id: currentRow?.id});
      hide();
      message.success('操作成功');
      return true;
    } catch (error: any) {
      hide();
      message.error('操作失败' + error.message);
      return false;
    }
  };
```

项目模块设计分析

接口信息表

id

name 接口名称

description 描述

url 接口地址

type 请求类型

requestHeader 请求头

responseHeader 响应头

status 接口状态 （0 关闭 1 开启）

isDelete

createTime

updateTime

```sql
-- 接口信息表
create table if not exists interface_info
(
    id              bigint auto_increment comment 'id' primary key,

    name            int comment '接口名称',
    description     varchar(512)                       null comment '接口描述',
    url             varchar(512)                       not null comment '接口地址',
    method          varchar(20)                        not null comment '请求类型',
    requestHeader   text                               null comment '请求头',
    responseHeader text                                null comment '响应头',
    status          tinyint(1) default 0               not null comment '接口状态（0 关闭 1 开启）',


    userId          bigint                             not null comment '创建人id',
    userName        varchar(20)                        not null comment '创建人姓名',
    createTime      datetime   default CURRENT_TIMESTAMP not null comment '创建时间',
    updateTime      datetime   default CURRENT_TIMESTAMP not null on update CURRENT_TIMESTAMP comment '更新时间',
    isDelete        tinyint    default 0               not null comment '是否删除'
) comment '接口信息表';
```

接口后端代码 - controller

```java
/**
 * 接口请求类型
 */
@RestController
@RequestMapping("/interfaceInfo")
@Slf4j
@Api(tags = "接口请求管理")
public class InterFaceInfoController {

    @Resource
    private InterfaceinfoService interfaceInfoService;


    @Resource
    private UserService userService;

    // region 增删改查

    /**
     * 创建
     */
    @PostMapping("/add")
    public BaseResponse<Long> addInterfaceInfo(@Validated @RequestBody
InterfaceInfoAddRequest addRequest, HttpServletRequest request) {

        interfaceInfoService.add(addRequest, request);

        return ResultUtils.success();
    }

    /**
     * 删除
     */
    @PostMapping("/delete")
    public BaseResponse<Boolean> deleteInterfaceInfo(@RequestBody
DeleteRequest deleteRequest, HttpServletRequest request) {
        if (deleteRequest == null || deleteRequest.getId() <= 0) {
            throw new BusinessException(ErrorCode.PARAMS_ERROR);
        }
        User user = userService.getLoginUser(request);
        long id = deleteRequest.getId();
        // 判断是否存在
        InterfaceInfo oldInterfaceInfo = interfaceInfoService.getById(id);
        ThrowUtils.throwIf(oldInterfaceInfo == null,
ErrorCode.NOT_FOUND_ERROR);
        // 仅本人或管理员可删除
        if (!oldInterfaceInfo.getUserId().equals(user.getId()) &&
!userService.isAdmin(request)) {
            throw new BusinessException(ErrorCode.NO_AUTH_ERROR);
        }
        boolean b = interfaceInfoService.removeById(id);
```

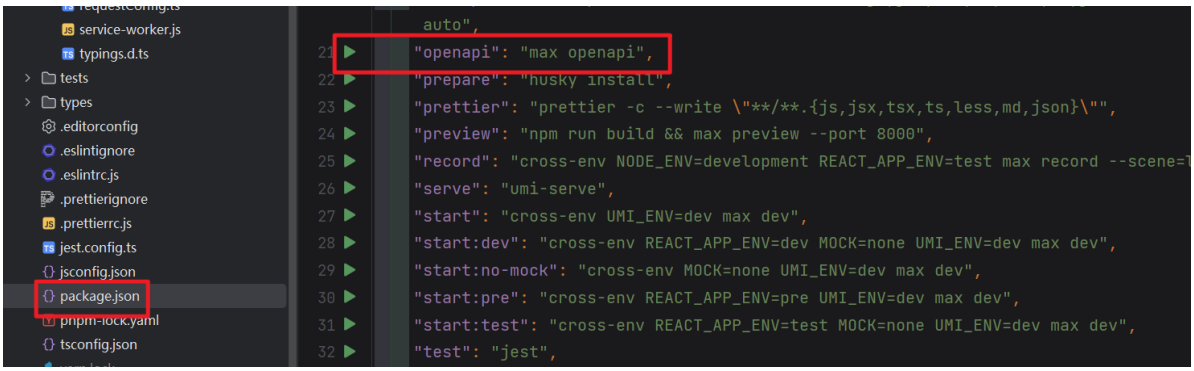
```
        return ResultUtils.success(b);
    }

    /**
     * 更新（仅管理员）
     */
    @PostMapping("/update")
    @AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
    public BaseResponse<Boolean> updateInterfaceInfo(@Validated
@RequestBody InterfaceInfoUpdateRequest interfaceInfoUpdateRequest) {
        if (interfaceInfoUpdateRequest == null ||
interfaceInfoUpdateRequest.getId() <= 0) {
            throw new BusinessException(ErrorCode.PARAMS_ERROR);
        }
        Boolean result =
interfaceInfoService.upt(interfaceInfoUpdateRequest);

        return ResultUtils.success(result);
    }

    /**
     * 根据 id 获取
     */
    @GetMapping("/get/vo")
    public BaseResponse<InterfaceInfoVo> getInterfaceInfoVOById(long id,
HttpServletRequest request) {
        if (id <= 0) {
            throw new BusinessException(ErrorCode.PARAMS_ERROR);
        }
        InterfaceInfo interfaceInfo = interfaceInfoService.getById(id);
        if (interfaceInfo == null) {
            throw new BusinessException(ErrorCode.NOT_FOUND_ERROR);
        }
        InterfaceInfoVo interfaceInfoVo =
BeanCopyUtil.copyBean(interfaceInfo, InterfaceInfoVo.class);
        return ResultUtils.success(interfaceInfoVo);
    }

    /**
     * 分页获取列表（封装类）
     */
    @PostMapping("/list/page/vo")
    public BaseResponse<PageVo> listInterfaceInfoVOByPage(@RequestBody
InterfaceInfoQueryRequest dto) {
        // 限制爬虫
        ThrowUtils.throwIf(dto.getPageSize() > 20, ErrorCode.PARAMS_ERROR);

        Page<InterfaceInfo> pageInfo =
interfaceInfoService.selectPage(dto);

        PageVo pageVo = new PageVo();
        pageVo.setCurrent(pageInfo.getCurrent());
        pageVo.setSize(pageInfo.getSize());
        pageVo.setReconds(pageInfo.getRecords());
        pageVo.setTotal(pageInfo.getTotal());
```

```
96
97          return ResultUtils.success(pageVo);
98      }
99
100 }
101
```

前端对接后端的代码 - openApi 生成


```
1 openAPI: [
2 {
3   requestLibPath: "import { request } from '@umijs/max'",
4   schemaPath: 'http://localhost:8102/api/v3/api-docs',
5   projectName: 'api-frontend',
6 },
```

> 点击即可

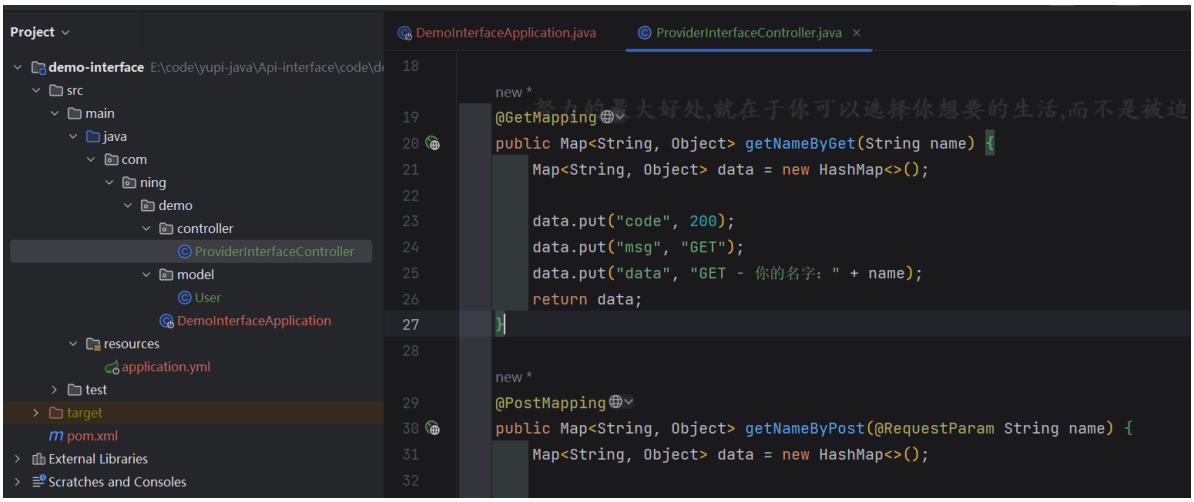


第二部分

模拟接口项目

主旨：模拟接口（类似第三方的接口，进行调用）

搭建一个新的项目



调用接口

几种HTTP 调用方式：

1. HttpClient
2. RestTemplate
3. 第三方库（OKHTTP、Hutool）

Hutool 使用

依赖引入

```
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.8.16</version>
</dependency>
```

> 使用

```
import cn.hutool.http.HttpRequest;
import cn.hutool.http.HttpUtil;
import cn.hutool.json.JSON;
import cn.hutool.json.JSONUtil;
import com.ning.demo.model.User;

import java.util.HashMap;

/**
```

```
 * @author NingWest
 * @date 2023/05/09 21:17
 */
public class MyClient {

    public String getMyName() {

        //可以单独传入http参数，这样参数会自动做URL编码，拼接在URL中
        HashMap<String, Object> paramMap = new HashMap<>();
        paramMap.put("name", "123");

        String result = HttpUtil.get("http://localhost:8103/api/demo",
paramMap);
        System.out.println(result);
        return result;
    }

    public String postMyName() {

        //可以单独传入http参数，这样参数会自动做URL编码，拼接在URL中
        HashMap<String, Object> paramMap = new HashMap<>();
        paramMap.put("name", "123");

        String result = HttpUtil.post("http://localhost:8103/api/demo",
paramMap);
        System.out.println(result);
        return result;
    }

    public String postJsonName(User user) {
        String jsonStr = JSONUtil.toJsonStr(user);
        return
HttpRequest.post("http://localhost:8103/api/demo/user").body(jsonStr).execut
e().body();
    }

    public static void main(String[] args) {
        MyClient myClient = new MyClient();

        String myName = myClient.getMyName();
        System.out.println(myName);
        String s = myClient.postMyName();
        System.out.println(s);
        User user = new User();
        user.setName("ning");
        String jsonUser = myClient.postJsonName(user);
        System.out.println(jsonUser);

    }
}
```

API签名认证

> 本质：

1. 签发签名
2. 使用签名（检验签名）

为什么需要？ 保证安全性。

> 怎么实现？

参数一：accessKey : 调用的标识 userA,userB

参数二：secretKey：密钥 （复杂、无序、无规律）该参数不妨到请求头中

（类似用户名和密码，区别：ak、sk是无状态的）

密钥不可直接在服务器之间传递，有可能会别拦截

参数三：sign

加密方式：对称加密、非对称加密、md5签名（不可解密）

签名认证算法与

用户参数 + 密钥 =》 签名算法 =》 不可解密的值

服务器用一模一样的参数和算法去生成签名，只要和用户传的一致，就表示一致。

注意问题：请求重放问题

参数4：加入nonce随机数，只能用一次

服务端要保存用过的随机数

参数5：加 timestamp时间戳，校验时间戳是否过期。

参数6：用户自己的请求参数

具体实现

用户表添加字段

先添加假数据，后期提供接口

> 校验参数的

```java
import cn.hutool.core.date.DateUtil;
import cn.hutool.core.util.StrUtil;
import com.ning.demo.constant.Constant;
import com.ning.demo.model.User;
import com.ning.demo.utils.Md5Utils;
import com.ning.demo.utils.SignUtils;

import javax.servlet.http.HttpServletRequest;
import java.util.Date;
import java.util.Map;

/**
 * @author NingWest
 * @date 2023/05/10 11:14
 */
public class CheckCore {

    public static boolean checkParams(User user, HttpServletRequest request)
{
        String sign = request.getHeader(Constant.SIGN);
        String nonce = request.getHeader(Constant.NONCE);
        String accessKey = request.getHeader(Constant.ACCESS_KEY);
        String timestamp = request.getHeader(Constant.TIMESTAMP);

        //先判断时间戳、随机数是否正确
        // 时间戳
        Date now = new Date();
        // 现在时间戳
        long nowTime = now.getTime();
        // 根据当前时间 获取两天前的时间戳
        long time = DateUtil.offsetDay(now, -2).getTime();
        // 如果传递过来的时间在这个范围内就证明是合法的时间请求
        if (StrUtil.isNotBlank(timestamp)) {
            // 转换为时间戳
            long timeStamp = Long.parseLong(timestamp);
            // 如果时间小于当前时间 并且大于 倒退2天 时间合理
            if (timeStamp >= nowTime || timeStamp < time) {
                return false;
            }
        }
        // todo nonce  此处的随机数 可以在生成的时候存入 redis 并携带过来唯一的UUID
根据 request获取
        if (StrUtil.isBlank(nonce)) {
            return false;
        }

        // todo 此处的 secretKey 应该是从数据库获取
        String newSign = Md5Utils.md5(user + "abcdefg" + accessKey);
        if (!newSign.equals(sign)) {
            return false;
        }

        // 最终无异常返回
```

```
        return true;
    }
}
```

客户端

```
import cn.hutool.core.util.RandomUtil;
import cn.hutool.http.HttpRequest;
import cn.hutool.http.HttpUtil;
import cn.hutool.json.JSON;
import cn.hutool.json.JSONUtil;
import com.ning.demo.model.User;
import com.ning.demo.utils.Md5Utils;
import com.ning.demo.utils.SignUtils;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

/**
 * @author NingWest
 * @date 2023/05/09 21:17
 */
public class MyClient {

    private String accessKey;

    private String secretKey;

    public MyClient() {
    }

    public MyClient(String accessKey, String secretKey) {
        this.accessKey = accessKey;
        this.secretKey = secretKey;
    }

    public String postJsonName(User user) {

        String jsonStr = JSONUtil.toJsonStr(user);
        return
                HttpRequest.post("http://localhost:8103/api/demo/user")
                        .addHeaders(SignUtils.getHeader(accessKey,
secretKey, user))
                        .body(jsonStr)
                        .execute()
                        .body();
    }


    public static void main(String[] args) {
        MyClient myClient = new MyClient("ning", "abcde");
        User user = new User();
```

```
        user.setName("宁西");
        String jsonUser = myClient.postJsonName(user);
        System.out.println(jsonUser);


    }

}
```

API签名认证是一个很灵活的设计，具体要有那些参数、参数名如何定义要根据常见来实现。（userId、appId、version、固定盐值。。。）

开发SDK（写简历）

开发者只需要关心调用那些接口、传递那些参数，就跟调用自己写的代码一样简单。

开发starter的好处：开发者引入之后，可以直接在application.yml中写配置，自动创建客户端

> 创建新的项目　api-client-sdk

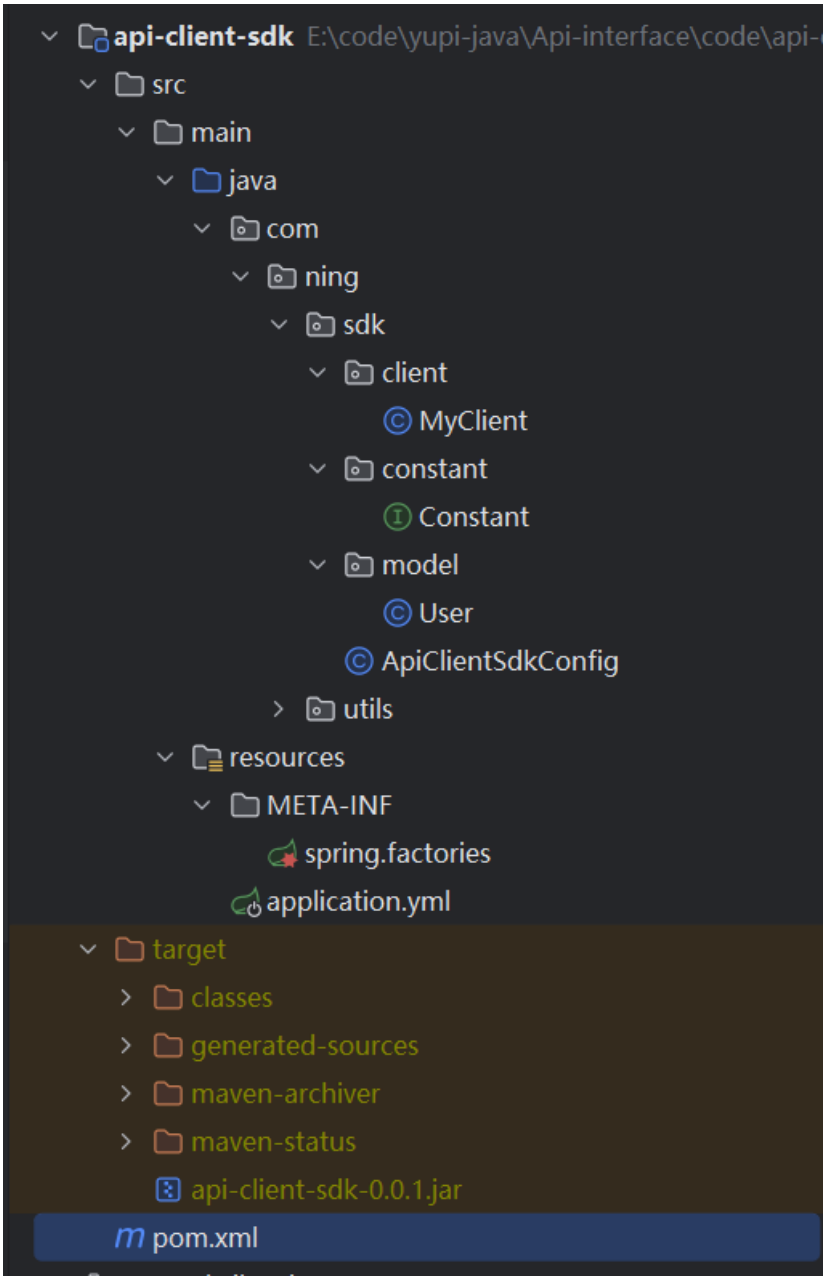
> 引入的依赖

spring-boot-configuration-processor　配置提示的依赖

> 去除的依赖 的maven打包依赖

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>
```

> 项目截图



> 其中 ApiClientSdkConfig

```java
import com.ning.sdk.client.MyClient;
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Data
@Configuration
@ComponentScan
@ConfigurationProperties("api.client")
public class ApiClientSdkConfig {

    private String accessKey;

    private String secretKey;

```

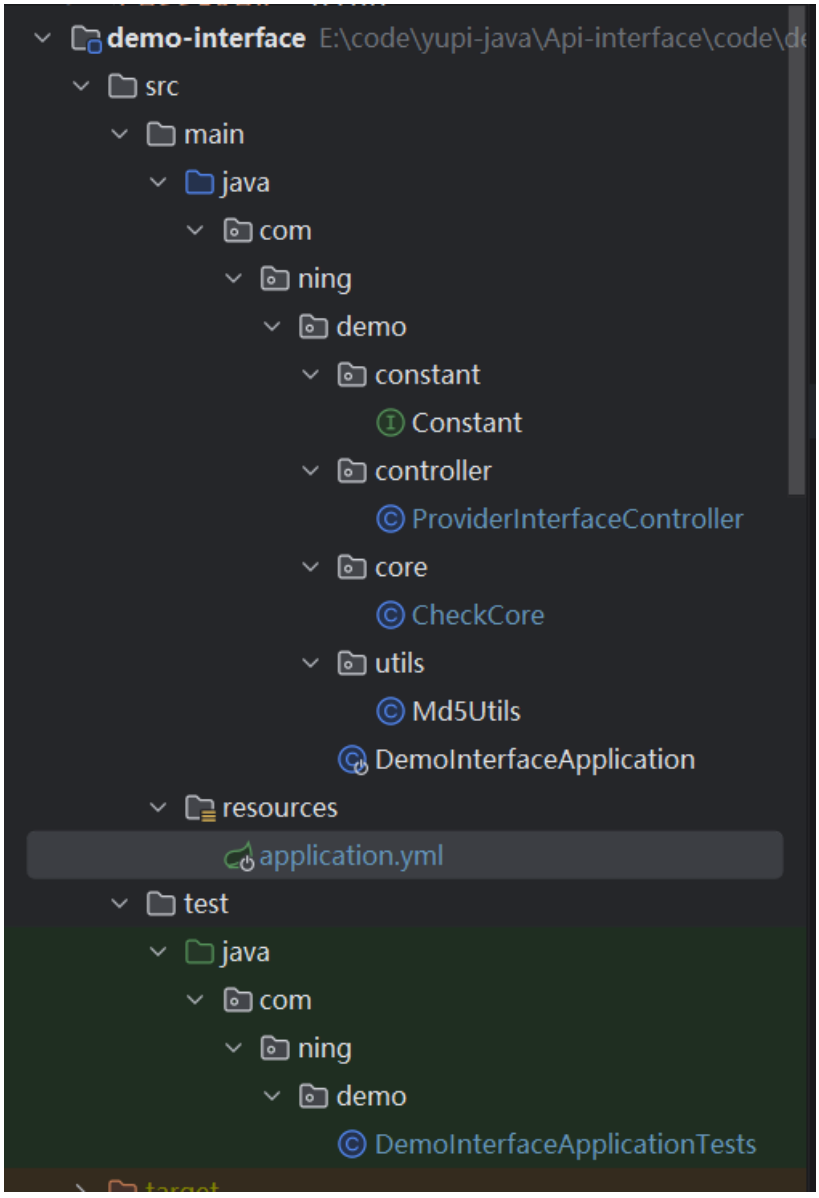
```
    @Bean
    public MyClient myClient() {
        return new MyClient(accessKey, secretKey);
    }
}
```

其中spring.factories 配置

```
# 自动加载
org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.ning.sdk.ApiClientSdkConfig
```

接口端配置

接口配置



添加依赖

```xml
<!-- 引入自己编写的starter -->
<dependency>
    <groupId>com.ning</groupId>
    <artifactId>api-client-sdk</artifactId>
    <version>0.0.1</version>
</dependency>
```

> application.yml

```yaml
# 配置
api:
  client:
    access-key: ning
    secret-key: abcdefg
```

> 测试内容

```java
import com.ning.sdk.client.MyClient;
import com.ning.sdk.model.User;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import javax.annotation.Resource;

@SpringBootTest
class DemoInterfaceApplicationTests {

    @Resource
    private MyClient myClient;

    @Test
    void contextLoads() {
        User user = new User();
        user.setName("ning");
        String myName = myClient.postJsonName(user);
        System.out.println(myName);
    }
}
```

> 结果

```
{"msg":"ERROR","code":500,"data":"ERROR 错误信息返回"}
```

第三部分

1. 开发接口发布 / 下线的功能（管理员）
2. 前端去浏览接口、查看接口文档、申请签名（注册）、在线调用（用户）
3. 统计用户调用接口的次数
4. 优化系统 - API网关

接口上线 / 下线

后台接口：

发布接口：（管理员）

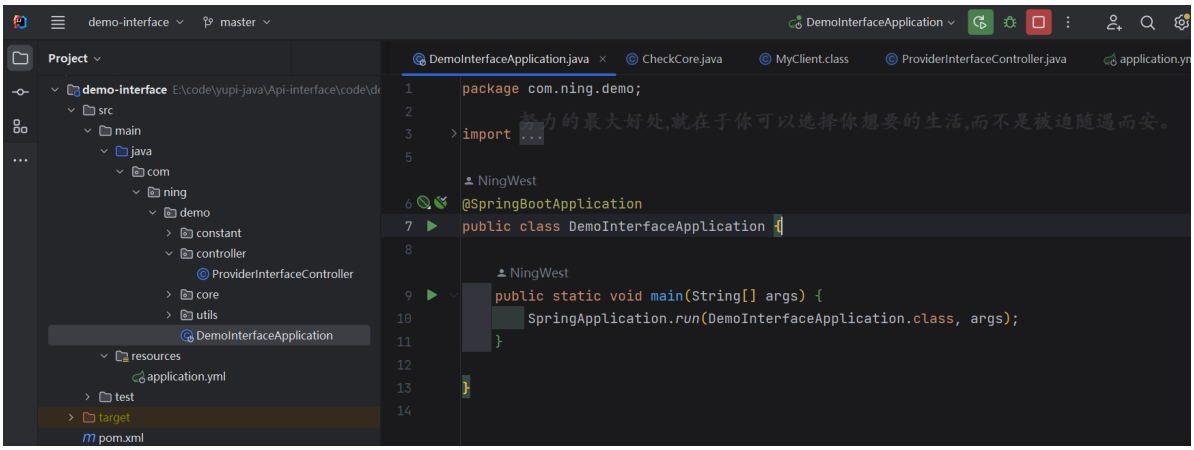
1. 校验该接口是否存在
2. 判断该接口是否可以调用
3. 修改接口数据库中的状态字段为1

下线接口：（管理员）

1. 校验该接口是否存在
2. 修改数据库中的状态字段为0

后端 实现

> 先启动第三方服务



> 在主服务中引入第三方依赖

```
<!-- 引入自己编写的依赖 -->
<dependency>
    <groupId>com.ning</groupId>
    <artifactId>api-client-sdk</artifactId>
    <version>0.0.1</version>
</dependency>
```

> 实现流程 部分代码

```
public void onlineInterfaceInfo(Long id) {
```

```
    InterfaceInfo interfaceInfo = this.getById(id);

    if (Objects.isNull(interfaceInfo)) {
        throw new BusinessException(ErrorCode.NO_HAVE_ERROR);
    }
    // 求地址
    String url = interfaceInfo.getUrl();
    if (StringUtils.isBlank(url)) {
        throw new BusinessException(ErrorCode.PARAMETER_ERROR);
    }
    // 引入第三方SDK 进行请求访问 注意：这里因为测试问题，没有真正调用  url地址 todo
    com.ning.sdk.model.User user = new com.ning.sdk.model.User();
    user.setName("ning test");

    String result = apiClient.postJsonName(user);
    if (StringUtils.isBlank(result)) {
        throw new BusinessException(ErrorCode.PARAMETER_ERROR);
    }
    BaseResponse bean = JSONUtil.toBean(result, BaseResponse.class);

    if (Objects.isNull(bean)) {
        throw new BusinessException(ErrorCode.PARAMETER_ERROR);
    }
    if (CommonConstant.ERROR_CODE == bean.getCode()) {
        throw new BusinessException(ErrorCode.NO_RETURN_ERROR);
    }

    // 接口状态（0 关闭 1 开启）  这里是 上线接口
    interfaceInfo.setStatus(1);

    if (!this.updateById(interfaceInfo)) {
        throw new BusinessException(ErrorCode.SYSTEM_ERROR);
    }

}
```

> 引入 配置文件配置

```
# 第三方服务的 密钥、密匙
# 配置
api:
  client:
    access-key: ning
    secret-key: abcdefg
```

前端 实现

> 调用 接口自动生成api 生成对应接口

```
record.status === 0 ? <a
  key="online"
  onClick={() => {
    onlineInterFace(record.id).then(r => {
      console.log(r)
    });
    setCurrentRow(record);
  }}
>
  上线
</a> : null,
```

> 实现方法

```
// 上线接口
  const onlineInterFace = async (id: any) => {

    try {
      await onlineInterfaceInfoUsingPOST({id});
      message.success('上线成功');
      // 删除完后刷新页面
      actionRef.current?.reload();
      return true;
    } catch (error: any) {
      message.error('上线失败：' + error.message);
      return false;
    }
  };
```

浏览接口/分页/详情

首页添加接口信息 + 分页

> 在 框架中隐藏

```
hideInMenu: true, // 菜单项显示
```

> 出现问题： 页面时间 少 8个消失的问题

解决办法：

链接数据库的时候：`?serverTimezone=Asia/Shanghai`

```
jdbc:mysql://localhost:3306/my_api_interface?serverTimezone=Asia/Shanghai
```

修改 `@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")`

添加 `timezone = "GMT+8"` 解决

> 分页组件 显示总条数

```
showTotal={(total) => `总条数： ${total}`}
```

> react 获取路由信息

```
let urlId = useParams()
```

一键复制，请求地址

> 添加依赖

```
yarn add copy-to-clipboard
```

> 代码部分

```
const copyCot = (cot: string) => {
   copy(cot);
   message.success('复制成功').then(r => {
     console.log(r)
   }); // 不需要提示 可注释
}

<Button onClick={() => copyCot(info.url as string)} type="primary">一键复制，请求地址</Button>
```

申请签名

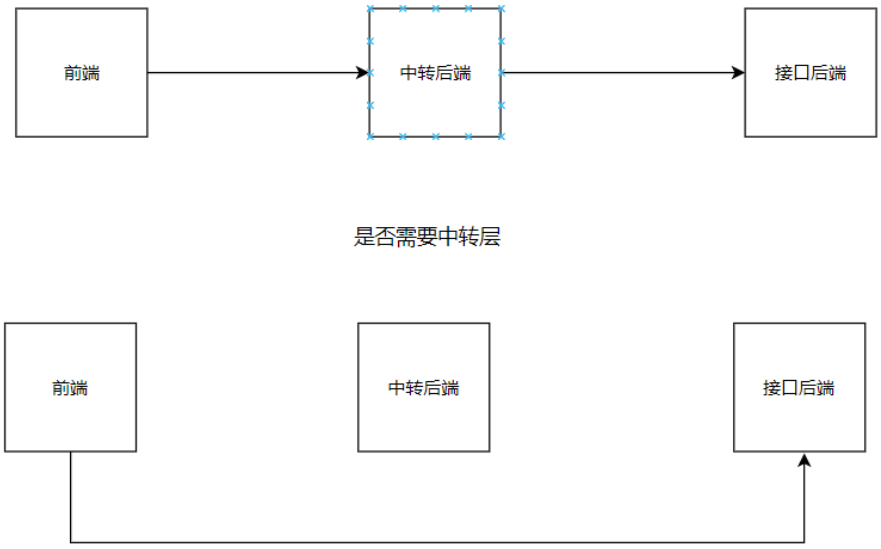
在注册的接口中进行。

使用 hutool 包的随机进行生成

```
// 生成 ak、sk
String ak = CommonConstant.API_PRE + RandomUtil.randomString(6) + "." + RandomUtil.randomString(10);
String sk = CommonConstant.API_PRE + RandomUtil.randomString(10) + "." + RandomUtil.randomString(10);
```

在线调用接口

> 流程图



后端

> 代码 后续优化

```
@Override
public Object invoke(InterfaceInvokeRequest invokeRequest,
HttpServletRequest request) {
    User loginUser = userService.getLoginUser(request);
    if (StringUtils.isBlank(loginUser.getAccessKey()) ||
StringUtils.isBlank(loginUser.getSecretKey())) {
        throw new BusinessException(ErrorCode.PARAMETER_ERROR);
    }
    MyClient myClient = new MyClient(loginUser.getAccessKey(),
loginUser.getSecretKey());
    // 格式化数据
    Gson gson = new Gson();
    com.ning.sdk.model.User user =
gson.fromJson(invokeRequest.getRequestParams(),
com.ning.sdk.model.User.class);
//        String url = invokeRequest.getUrl();  请求地址
    // 发送请求 todo 有待改进
    String resp = myClient.postJsonName(user);
    return gson.fromJson(resp, Response.class);
}
```

前端

```
const onFinish = async (values: string) => {
    // 骨架平 出现
    setResLoading(true)
    // @ts-ignore
    let {params} = values
    // 校验字符串
    if (isNil(params)) {
      message.error('缺失请求参数').then(r => {
        console.log(r)
      });
      return
    }
    // 请求接口
    let result = await invokeInterfaceInfoUsingPOST({
      id: info.id,
      requestParams: params,
      url: info.url
    })
    let {code, data} = result;
    if (code === 0) {
      message.success("发送请求成功")
      // @ts-ignore
      setDataInfo(data?.data)
      setTimeout(() => {
        // 请求成功，骨架平隐藏
        setResLoading(false)
      }, 200)
    }
```

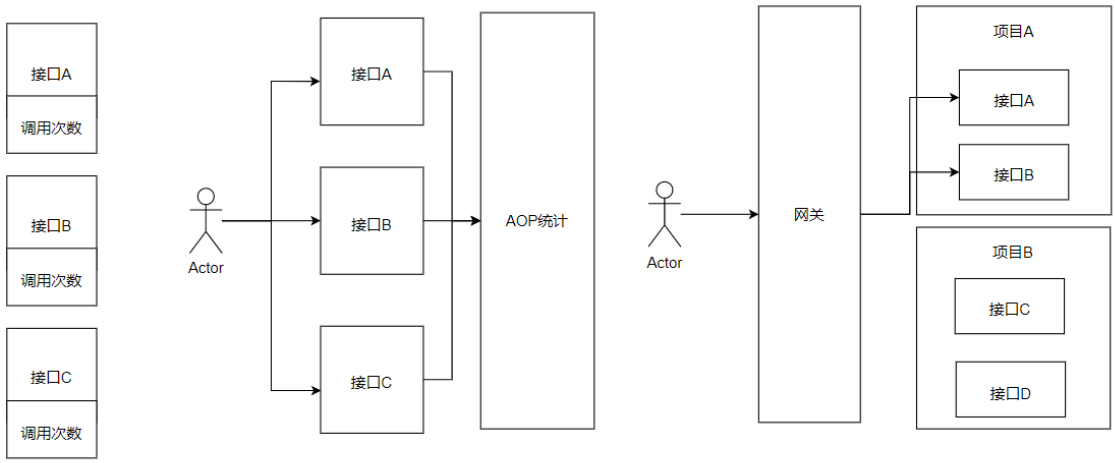
详细 代码仓库

第四部分

1. 开发接口调用次数的统计
2. 优化整个系统的架构（API网关）
    - 网关是什么?
    - 网关的作用?
    - 网关的应用场景?
    - 结合业务去应用网关

> 网关图片

网关



使用 AOP 切面的优点：独立于接口，在每个接口调用后统计次数 + 1

AOP 切面的缺点：只存在于单个项目中，如果每个团队都要开发自己的模拟接口，那么都要写一个切面

网关

什么是网关？理解成火车站的检票口，**统一**去检票

网关的优点：统一去进行一些操作、处理一些问题

作用

1. 路由
2. 鉴权
3. 跨域
4. 流量染色
5. 访问控制
6. 统一业务处理（缓存）
7. 发布控制
8. 负载均衡
9. 接口保护
    - 限制请求
    - 信息脱敏
    - 降级（熔断）
    - 限流
    - 超时时间
10. 统一日志
11. 统一文档

路由：

起到转发的作用，比如有接口A和接口B，网关会记录这些信息，根据用户访问的地址和参数，转发请求到对应的接口（服务器/集群）

/a ==> 接口A

/b ==> 接口B

负载均衡

在路由的基础上

/c => 服务A /集群A （随机转发到其中的某一个机器上）

鉴权

判断用户是否有权限进行操作，无论访问什么接口，统一去判断权限，不用重复写。

统一处理跨域

网关统一处理跨域，不用单一处理

统一业务处理

把每个项目中都要做的通用逻辑放在上层（网关），统一处理。

访问控制

黑白名单，比如限制DDOS IP

发布控制

灰度发布 理解：两个接口 接口A实现发布的，后续要替换接口A为V2版本，不清楚V2版本是否稳定，指定一部分流量流向V2接口，其他的还是流向A接口，再慢慢调整比例。

流量染色

给请求（流量）添加一些标识，一般是在请求头中添加，新的请求信息

接口保护

- 限制请求
- 信息脱敏
- 降级（熔断）
- 限流
- 超时时间

统一日志

统一的请求，响应信息记录

统一文档

将下游项目的文档进行聚合，在一个页面统一查看

网关的分类

1. 全局网关（接入层网关）：作用 是负载均衡、请求日志等，不和业务逻辑绑定。
2. 业务网关（微服务网关）：会有一些业务逻辑，作用是将请求转发到不同的业务 / 项目 / 服务

实现

1. Nginx（全局网关）、Kong网关（API网关）
2. Spring Cloud Gateway (取代了Zuul)性能高、可以用java代码书写业务逻辑

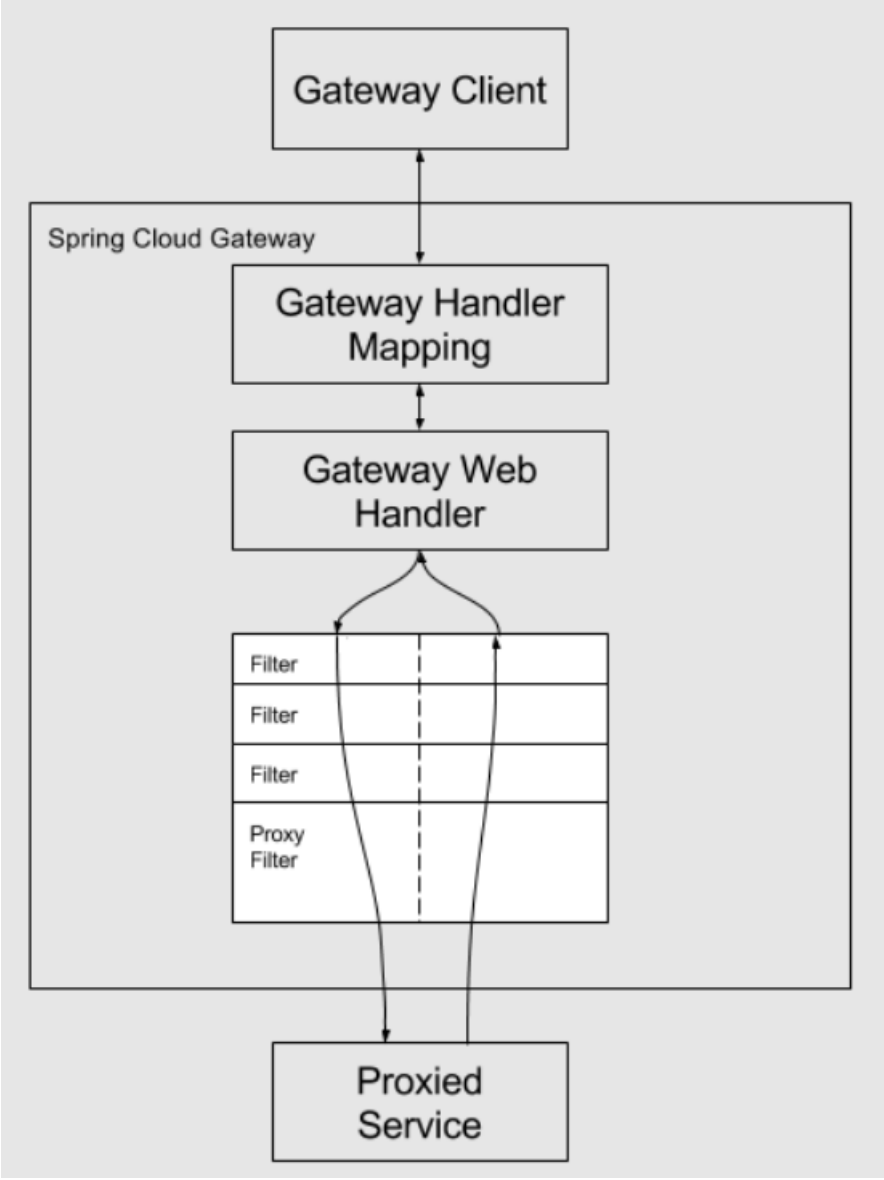
> 引入依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

> 配置类

```yaml
server:
  port: 8090

spring:
  cloud:
    gateway:
      routes:
        - id: yuPi
          uri: http://yupi.icu
          predicates:
            - Path=/
            - After=2023-01-20T17:42:47.789-07:00[America/Denver]
```

> 工作结构图



> 错误记录

```
 springboot2.7.2     Unable to find GatewayFilterFactory with name CircuitBreaker 报错
```

> 解决办法 将原来的 **resilience4j 替换为 下面的依赖**

```
<!-- 熔断   此处修改版版本才可以-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
    <version>2.2.10.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-reactor-resilience4j</artifactId>
    <version>2.1.3</version>
</dependency>
```

业务逻辑

1. 用户发送请求到网关
2. 请求日志
3. （黑白名单）
4. 用户鉴权（判断 ak、sk是否合法）
5. 请求的模拟接口是否存在?
6. 请求转发，调用模拟接口
7. 相应日志
8. 调用成功，接口调用次数+1
9. 调用失败，返回一个规范的错误码

具体实现

> yml 配置

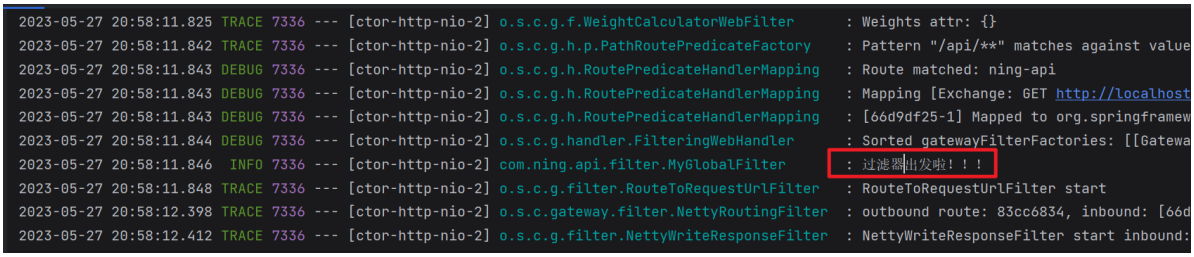
```
spring:
  cloud:
    gateway:
      routes:
        - id: ning-api
          uri: http://localhost:8103
          predicates:
            - Path=/api/**
```

成功截图


```
{
    "msg":  "GET",
    "code":  200,
    "data":  "GET - 你的名字：123123"
}
```


配置全局过滤器

```java
import lombok.extern.slf4j.Slf4j;
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;

/**
 * @author NingWest
 * @date 2023/05/27 20:55
 */
@Slf4j
@Component
public class MyGlobalFilter implements GlobalFilter, Ordered {
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

        log.info("过滤器出发啦！！！");
        return chain.filter(exchange);
    }

    @Override
    public int getOrder() {
        return -1;
    }
}
```



第五部分

> 实现gateway 日志输出

```java
import cn.hutool.core.date.DateUtil;
import cn.hutool.core.util.StrUtil;
import cn.hutool.http.HttpUtil;
import com.ning.api.constant.Constant;
import com.ning.api.utils.Md5Utils;
import lombok.extern.slf4j.Slf4j;
import org.reactivestreams.Publisher;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.core.Ordered;
import org.springframework.core.io.buffer.DataBuffer;
import org.springframework.core.io.buffer.DataBufferFactory;
import org.springframework.core.io.buffer.DataBufferUtils;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.server.reactive.ServerHttpRequest;
import org.springframework.http.server.reactive.ServerHttpResponse;
import
org.springframework.http.server.reactive.ServerHttpResponseDecorator;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.nio.charset.StandardCharsets;
import java.util.*;

/**
 * @author NingWest
 * @date 2023/05/27 20:55
 */
@Slf4j
@Component
@ConfigurationProperties(prefix = "my")
public class MyGlobalFilter implements GlobalFilter, Ordered {

    /*-- 获取配置文件中的集合  方便设置白名单 --*/
    private List<String> whiteList;

```

```
    public List<String> getWhiteList() {
        return whiteList;
    }

    public void setWhiteList(List<String> whiteList) {
        this.whiteList = whiteList;
    }

    /*-- 结束 --*/
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        //1. 请求日志
        ServerHttpRequest request = exchange.getRequest();
        log.info("请求地址：[{}]", request.getPath().value());
        log.info("请求方法：[{}]", request.getMethod());
        // 请求地址
        String address = Objects.requireNonNull(request.getLocalAddress()).getHostString();
        log.info("请求来源地址：[{}]", address);
        //2. 黑白名单  只有127.0.0.1的才能访问
        ServerHttpResponse response = exchange.getResponse();
        if (!whiteList.contains(address)) {
            response.setStatusCode(HttpStatus.FORBIDDEN);
            // 返回
            return response.setComplete();
        }
        //3. 用户鉴权（判断 ak、sk是否合法）
        boolean isOk = checkMethod(request);
        if (!isOk) {
            response.setStatusCode(HttpStatus.FORBIDDEN);
            // 返回
            return response.setComplete();
        }
        return handleResponse(exchange, chain);
    }

    /**
     * 处理响应
     */
    public Mono<Void> handleResponse(ServerWebExchange exchange, GatewayFilterChain chain) {
        try {
            ServerHttpResponse originalResponse = exchange.getResponse();
            // 缓存数据的工厂
            DataBufferFactory bufferFactory = originalResponse.bufferFactory();
            // 拿到响应码
            HttpStatus statusCode = originalResponse.getStatusCode();
            if (statusCode == HttpStatus.OK) {
                // 装饰，增强能力
                ServerHttpResponseDecorator decoratedResponse = new ServerHttpResponseDecorator(originalResponse) {
                    // 等调用完转发的接口后才会执行
                    @Override
```

```
                    public Mono<Void> writeWith(Publisher<? extends DataBuffer> body) {
                        log.info("body instanceof Flux: {}", (body instanceof Flux));
                        if (body instanceof Flux) {
                            Flux<? extends DataBuffer> fluxBody = Flux.from(body);
                            // 往返回值里写数据
                            // 拼接字符串
                            return super.writeWith(
                                    fluxBody.map(dataBuffer -> {
                                        try {
                                            // 7. 调用成功，接口调用次数 + 1 invokeCount
                                            Map<String, Object> params = new HashMap<>();
                                            params.put("userInterFaceId", 14);
                                            // 相应结果
                                            String result = HttpUtil.get("http://localhost:8102/userInterfaceInfo/minusOne", params);
                                            System.out.println(result);

                                        } catch (Exception e) {
                                            log.error("invokeCount error", e);
                                        }
                                        byte[] content = new byte[dataBuffer.readableByteCount()];
                                        dataBuffer.read(content);
                                        DataBufferUtils.release(dataBuffer);//释放掉内存
                                        // 构建日志
                                        StringBuilder sb2 = new StringBuilder(200);
                                        List<Object> rspArgs = new ArrayList<>();
                                        rspArgs.add(originalResponse.getStatusCode());
                                        String data = new String(content, StandardCharsets.UTF_8); //data
                                        sb2.append(data);
                                        // 打印日志
                                        log.info("响应结果: " + data);
                                        return bufferFactory.wrap(content);
                                    }));
                        } else {
                            // 8. 调用失败，返回一个规范的错误码
                            log.error("<--- {} 响应code异常", getStatusCode());
                        }
                        return super.writeWith(body);
                    }
                };
                // 设置 response 对象为装饰过的
```

```
                return chain.filter(exchange.mutate().response(decoratedResponse).build());
            }
            return chain.filter(exchange); // 降级处理返回数据
        } catch (Exception e) {
            log.error("网关处理响应异常" + e);
            return chain.filter(exchange);
        }
    }


    /**
     * 请求校验
     */
    private boolean checkMethod(ServerHttpRequest request) {
        // 获取请求头
        HttpHeaders headers = request.getHeaders();
        String sign = headers.getFirst(Constant.SIGN);
        String nonce = headers.getFirst(Constant.NONCE);
        String accessKey = headers.getFirst(Constant.ACCESS_KEY);
        String timestamp = headers.getFirst(Constant.TIMESTAMP);
        String params = headers.getFirst(Constant.DATA);

        //先判断时间戳、随机数是否正确
        // 时间戳
        Date now = new Date();
        // 现在时间戳
        long nowTime = now.getTime();
        // 根据当前时间 获取5分钟的时间戳
        long time = DateUtil.offsetMinute(now, -5).getTime();
        // 如果传递过来的时间在这个范围内就证明是合法的时间请求
        if (StrUtil.isNotBlank(timestamp)) {
            // 转换为时间戳
            long timeStamp = Long.parseLong(Objects.requireNonNull(timestamp));
            // 如果时间小于当前时间 并且大于 倒退2天 时间合理
            if (timeStamp >= nowTime || timeStamp < time) {
                return false;
            }
        }
        // todo nonce  此处的随机数 可以在生成的时候存入 redis 并携带过来唯一的UUID 根据 request获取
        if (StrUtil.isBlank(nonce)) {
            return false;
        }

        // todo 此处的 secretKey 应该是从数据库获取
        String newSign = Md5Utils.md5(params + "abcdefg" + accessKey);
        if (!Objects.equals(newSign, sign)) {
            return false;
        }

        // 最终无异常返回
        return true;
    }
```

```
182
183     @Override
184     public int getOrder() {
185         return -1;
186     }
187 }
188
```

第六部分

计划

1. 补充完整的网关业务逻辑（怎么去操作数据库、怎么复用之前的方法？RPC）
2. 完善系统、开发一个监控统计功能

网关业务逻辑

问题：网关项目标价纯净，没有操作数据库的包、并且还要调用我们之前写过的代码？复制粘贴维护麻烦

理想：直接请求到其他项目的方法

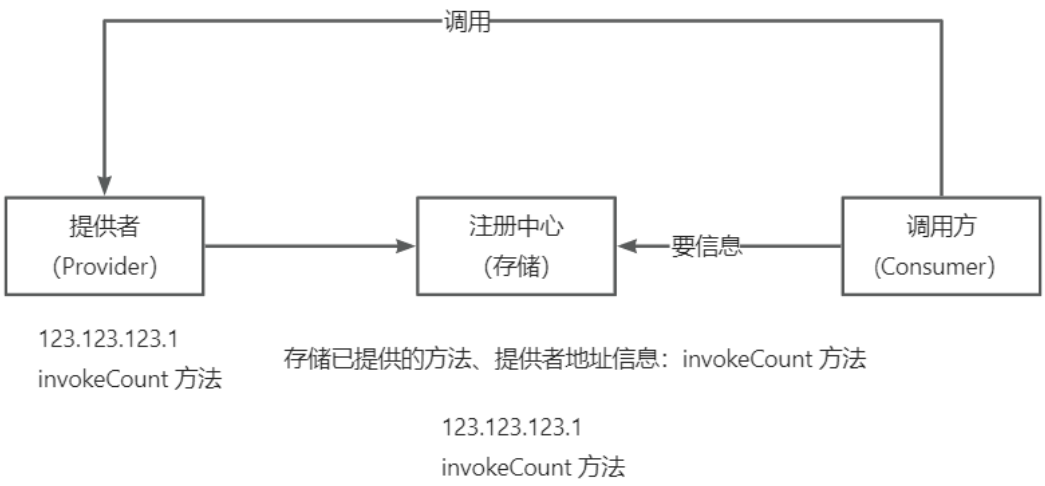
怎末调用其他项目的方法

1. 复制代码和依赖、环境
2. HTTP请求（提供一个接口，供其他项目调用）
3. RPC
4. 把公共的代码打个jar包，其他项目去引用（客户端SDK）

HTTP请求怎么调用

1. 提供方提供一个接口（地址、请求方法、参数、返回值）
2. 调用方使用HTTP Client 之类的代码包去发送HTTP请求

RPC



作用：像调用本地方法一样调用远程方法。

1. 对开发这更透明，减少了很多的沟通成本。
2. RPC向远程服务器发送请求，未必要使用HTTP协议，比如还可以用TCP/IP，新能更高（内部服务更适用）。

跑通Dubbo例子

> 地址

```
https://github.com/apache/dubbo-samples/tree/master/1-basic/dubbo-samples-spring-boot
```

> 出现问题

这个版本的dubbo中并没有 内嵌的 zookeeper ， 自己按一个win版本的

> 下载地址

[apache-zookeeper-3.8.0-bin.tar.gz]

> 使用教程

主要修改配置

```
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=D:/environment/zookeeper/apache-zookeeper-3.8.0-bin/data
dataLogDir=D:/environment/zookeeper/apache-zookeeper-3.8.0-bin/log
# the port at which the clients will connect
```

地址博客

> 遇到问题　**打开 zkServer.cmd 闪退问题**

一个 zkServer.cmd　一个zkClient.cmd

```
java "-Dzookeeper.log.dir=%ZOO_LOG_DIR%"
```

```
16
17  setlocal
18  call "%~dp0zkEnv.cmd"
19
20  set ZOOMAIN=org.apache.zookeeper.server.quorum.QuorumPeerMain
21  set ZOO_LOG_FILE=zookeeper-%USERNAME%-server-%COMPUTERNAME%.log
22
23  echo on
24  java "-Dzookeeper.log.dir=%ZOO_LOG_DIR%" "-Dzookeeper.log.file=%ZOO_LOG_FILE%" "-XX
25
26  endlocal
27
```

修改完毕之后，启动即可

项目集成 Dubbo

启动本地的zookeeper

zkServer.cmd

服务主体项目

调用数据库的主体

> 引入依赖

```
<!-- 引入 dubbo -->
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
    <version>3.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-dependencies-zookeeper-curator5</artifactId>
    <version>3.2.0</version>
    <exclusions>
        <exclusion>
            <artifactId>slf4j-reload4j</artifactId>
            <groupId>org.slf4j</groupId>
        </exclusion>
```

```
    </exclusions>
</dependency>
```

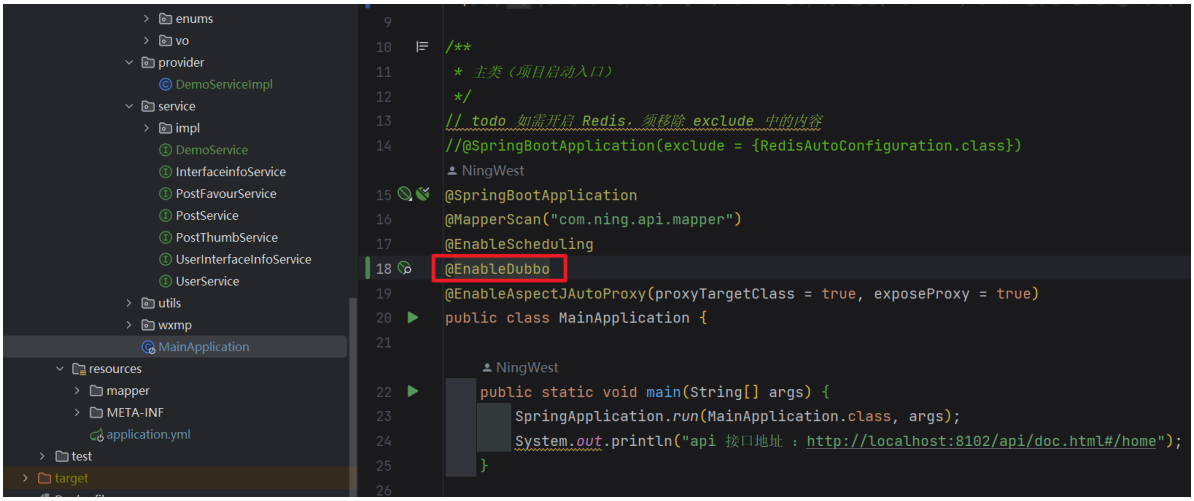
配置

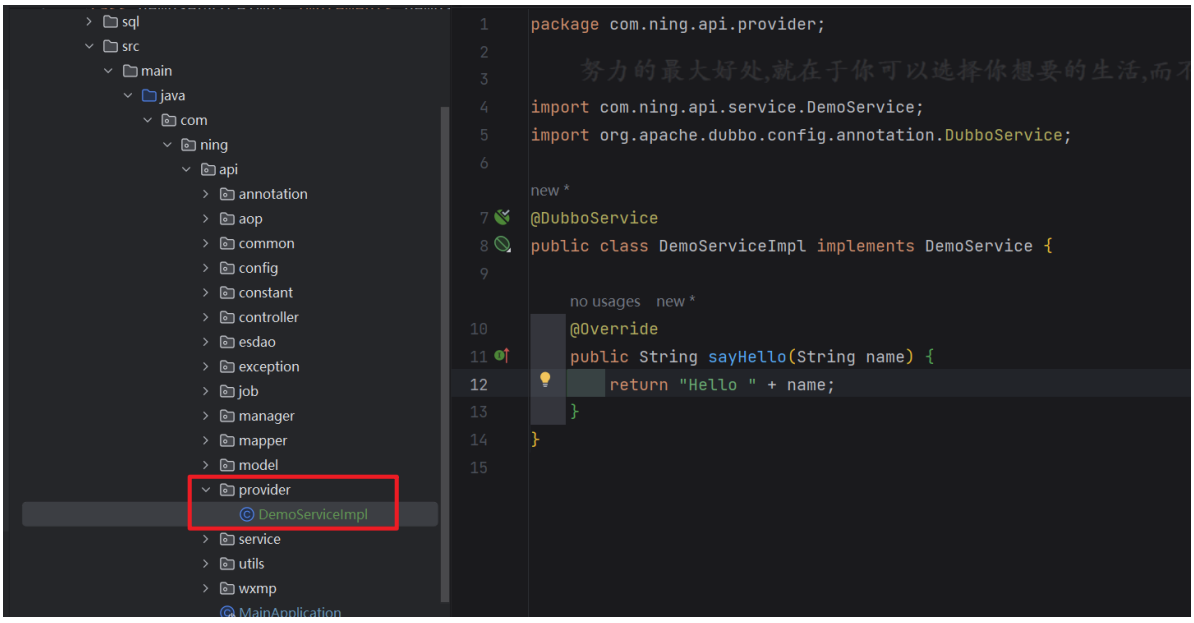
```yaml
# dubbo
dubbo:
  application:
    name: api-interface-provider
  protocol:
    name: dubbo
    port: -1
  registry:
    address: zookeeper://127.0.0.1:2181
```

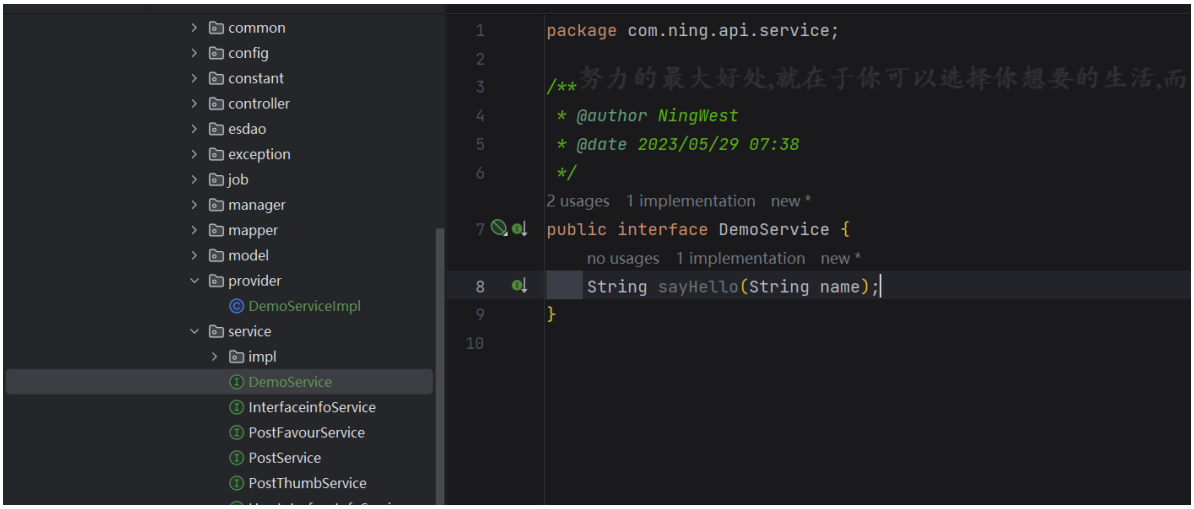
两部分代码

1. 与消费者端相同的接口
2. 因为这是服务的提供者，要写具体的服务提供实现

提供者 注解





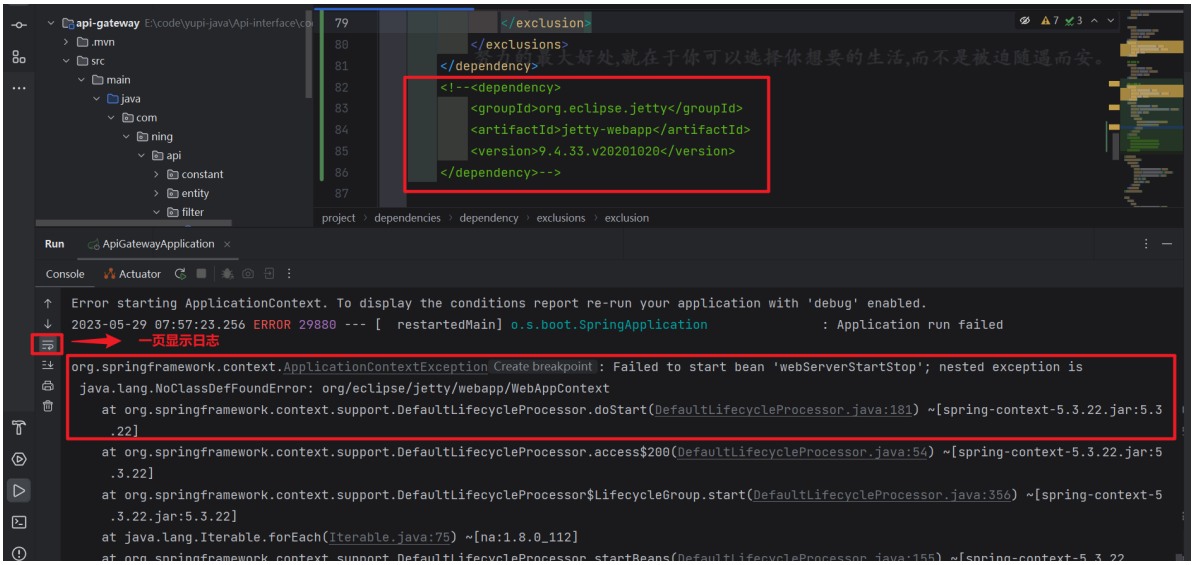


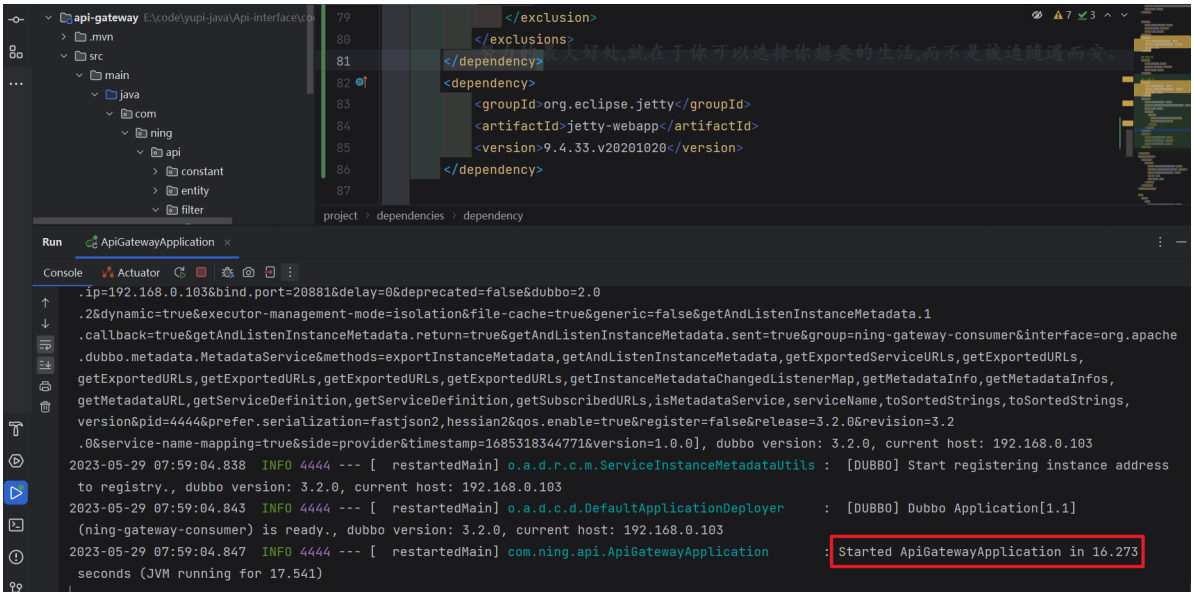
Gateway网关集成Dubbo

> 依赖

```xml
<!-- 引入 dubbo -->
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
    <version>3.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-dependencies-zookeeper-curator5</artifactId>
    <version>3.2.0</version>
    <exclusions>
        <exclusion>
            <artifactId>slf4j-reload4j</artifactId>
            <groupId>org.slf4j</groupId>
        </exclusion>
    </exclusions>
</dependency>
<!-- 这个依赖不引入会报错 ！！ -->
<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-webapp</artifactId>
    <version>9.4.33.v20201020</version>
</dependency>
```

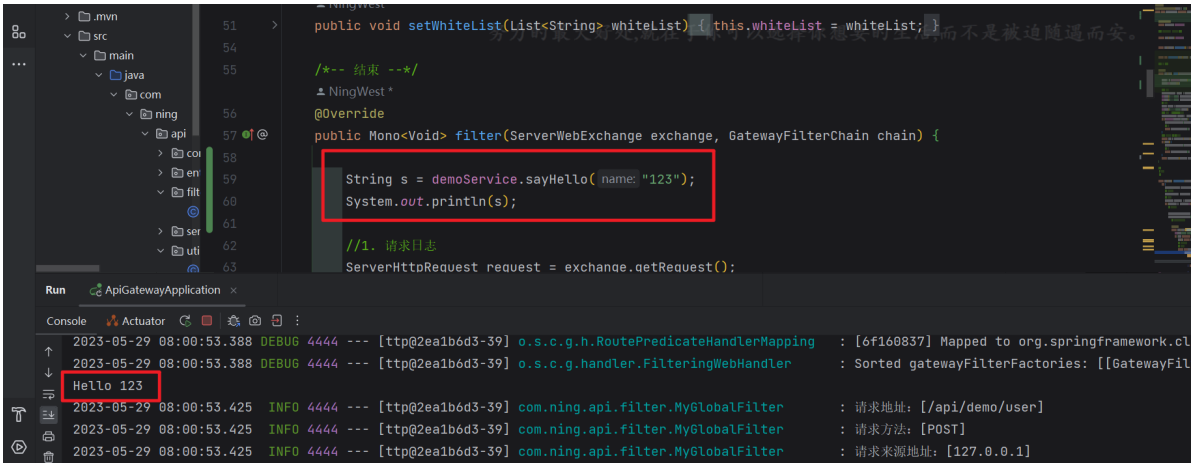
> 报错





> 消费者注解

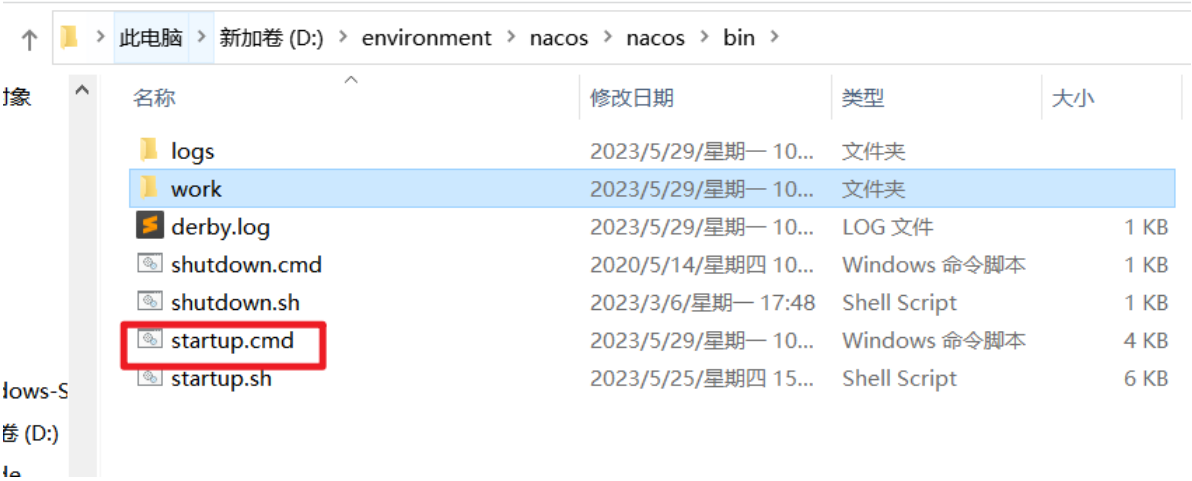
> 接收结果成功



win 安装nacos

[下载Win版本]

> 使用



> cmd 方式打开，输入命令

```
startup.cmd -m standalone
```

> 出现报错

```
D:\environment\nacos\nacos\bin>startup.cmd -m standalone
Please set the JAVA_HOME variable in your environment, We need java(x64)! jdk8 or later is better!

D:\environment\nacos\nacos\bin>
```

java环境也有

```
D:\environment\nacos\nacos\bin>java -version
java version "1.8.0_112"
Java(TM) SE Runtime Environment (build 1.8.0_112-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.112-b15, mixed mode)

D:\environment\nacos\nacos\bin>
```

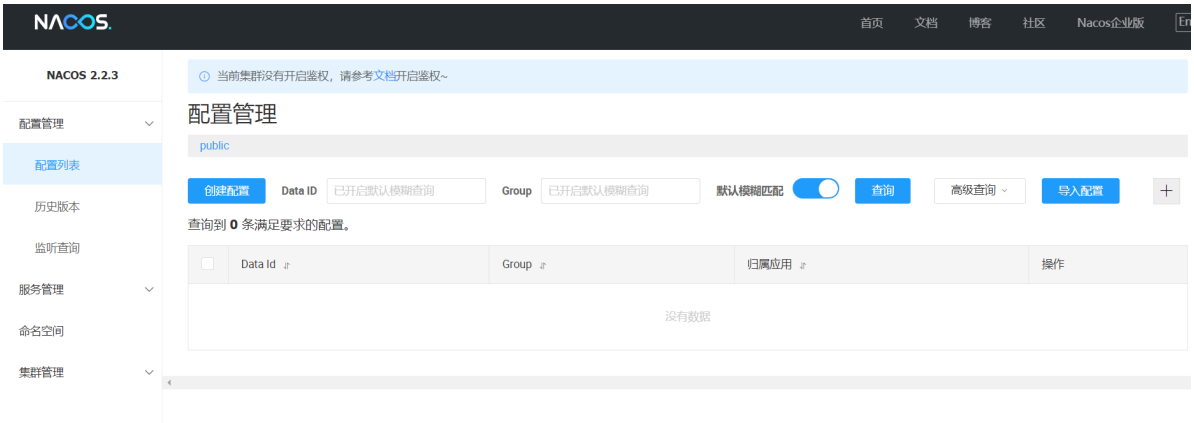
> 解决办法，打开 **startup.cmd** 修改其中的配置

```
set JAVA_HOME="**\JDK8"
```

```
rem      http://www.apache.org/licenses/LICENSE-2.0
rem
rem Unless required by applicable law or agreed to in writing, so
rem distributed under the License is distributed on an "AS IS" BA
rem WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
rem See the License for the specific language governing permissio
rem limitations under the License.
set JAVA_HOME="[redacted]\JDK8"
if not exist "%JAVA_HOME%\bin\java.exe" echo Please set the JAVA_
set "JAVA=%JAVA_HOME%\bin\java.exe"

setlocal enabledelayedexpansion

```

改成自己的jdk路径

> 成功



集成Nacos

> 提供者、消费者 配置

```yaml
# 更换为 nacos 为注册中心
dubbo:
  application:
    name: ning-gateway-provider
  protocol:
    name: dubbo
    port: -1
  registry:
    address: nacos://localhost:8848
```

第七部分

> 主要内容

1. 开发抽象功能服务（自己先不抽离了）
2. 实现网关核心业务流程 √
3. 开发管理员接口分析功能
4. 上线分析和扩展

填写todo

1. 查询数据库 完成 真正的 ak、sk 使用
2. 检测调用接口是否存在于数据库 dubbo调用
3. 调用成功 接口调用次数 leftNum 减一

统计分析功能

需求：生成图表，分析接口的调用情况，饼状图。

实现

前端：

- Echarts https://echarts.apache.org/examples/zh/index.html
- AntV: https://antv.vision/zh
- BizCharts: https://bizcharts.taobao.com/

React 使用 Echarts

https://github.com/hustcc/echarts-for-react

> 安装 **两个都需要安装**

```
npm install --save echarts-for-react

npm install echarts
```

> 代码

```
import ReactECharts from 'echarts-for-react';

// render echarts option.
<ReactECharts option={this.getOption()} />
```

后端

SQL语句：

```
SELECT interfaceInfoId, sum(totalNum) as totalNum
        FROM `user_interface_info`
        GROUP BY interfaceInfoId
        ORDER BY totalNum DESC
        LIMIT 10
```

页面修改

> 取出 登陆页面右上角 的 国际化图表

```
<Lang/>

const Lang = () => {
  const langClassName = useEmotionCss(({token}) => {
    return {
      width: 42,
      height: 42,
      lineHeight: '42px',
      position: 'fixed',
      right: 16,
      borderRadius: token.borderRadius,
      ':hover': {
```

```
        backgroundColor: token.colorBgTextHover,
      },
    };
  });

  return (
    <div className={langClassName} data-lang>
      {/*{SelectLang && <SelectLang/>}*/}
    </div>
  );
};

```

找打上述的代码 删除这两部分

后续补充实现

每一期结束后就写了 todo 鱼皮哥后期一 一解决了，也自己提了一些。

修改SDK中的 验证问题 替换为真是数据库的数据信息 √

修改SDK中的地址，为传输地址 √

其中的 todo 后期可以补充 √

自己实现签名等等的实现 √

todo 如何实现不同类型，不同参数，sdk调用不同接口?

todo 自己搭建的 minio 图片服务器 也可以整理出 starter ？

+

鱼皮哥的扩展？