接口操作文档

目录

[接口操作文档 1](#_Toc42788136)

[1. 概述 2](#_Toc42788137)

[2. 首页 2](#_Toc42788138)

[3. 接口管理 2](#_Toc42788139)

[查询 2](#_Toc42788140)

[新增 2](#_Toc42788141)

[删除 3](#_Toc42788142)

[4. 接口日志 3](#_Toc42788143)

[5. Charts 4](#_Toc42788144)

[6. 开发帮助 4](#_Toc42788145)

[HTTP 4](#_Toc42788146)

[POST 4](#_Toc42788147)

[GET 5](#_Toc42788148)

[WEBSERVICE 7](#_Toc42788149)

[字符串 7](#_Toc42788150)

[对象 7](#_Toc42788151)

[7. 协议转发 8](#_Toc42788152)

1. 概述

接口管理平台作用于外部系统接口接入、参数格式转换、接口数据整合及非业务逻辑操作。现阶段支持协议HTTP（get,post）,WEBSERVICE(字符串、对象)四种方式，平台提供高扩展性接口，根据扩展方式就可以实现消息队列等其他接口方式。

1. 首页

统计当前接口执行情况，

比如：

* 当天总接口失败次数
* 当天生产者调用失败次数
* 当天消费者调用失败次数
* 当天接口调用数前8名及成功统计
* 项目统计

1. 接口管理

查询



新增



**接口编码：**接口唯一标识，根据接口编码获取接口配置信息。

**接口名称：**接口描述

**状态：**启用/禁用,只有接口为启用才可以调用。

**是否重试：**是/否 ，当接口发生网络异常时就会进行接口重试。次数会根据重试次数数量决定，单次接口调用超时时间根据配置获取。

**生产者名称：**接口提供者名称

**生产者地址：**接口提供者发布地址，会根据这个地址去请求

**生产者方法：**接口请求的方法名

**消费者名称：**谁调用这个接口

**消费者方法名：**调用这个的方法名，主要用于问题定位时方便查看

**消费者类名：**调用这个的类名，主要用于问题定位时方便查看

**IP访问设置：**配置的IP才可以进行访问，支持通配

**生产者参数格式：**接口提供者的参数格式，主要用于查看接口参数

**消费者参数格式：**调用者参数格式，主要用于查看接口参数

删除

移除接口，将不能在进行调用和监控

1. 接口日志

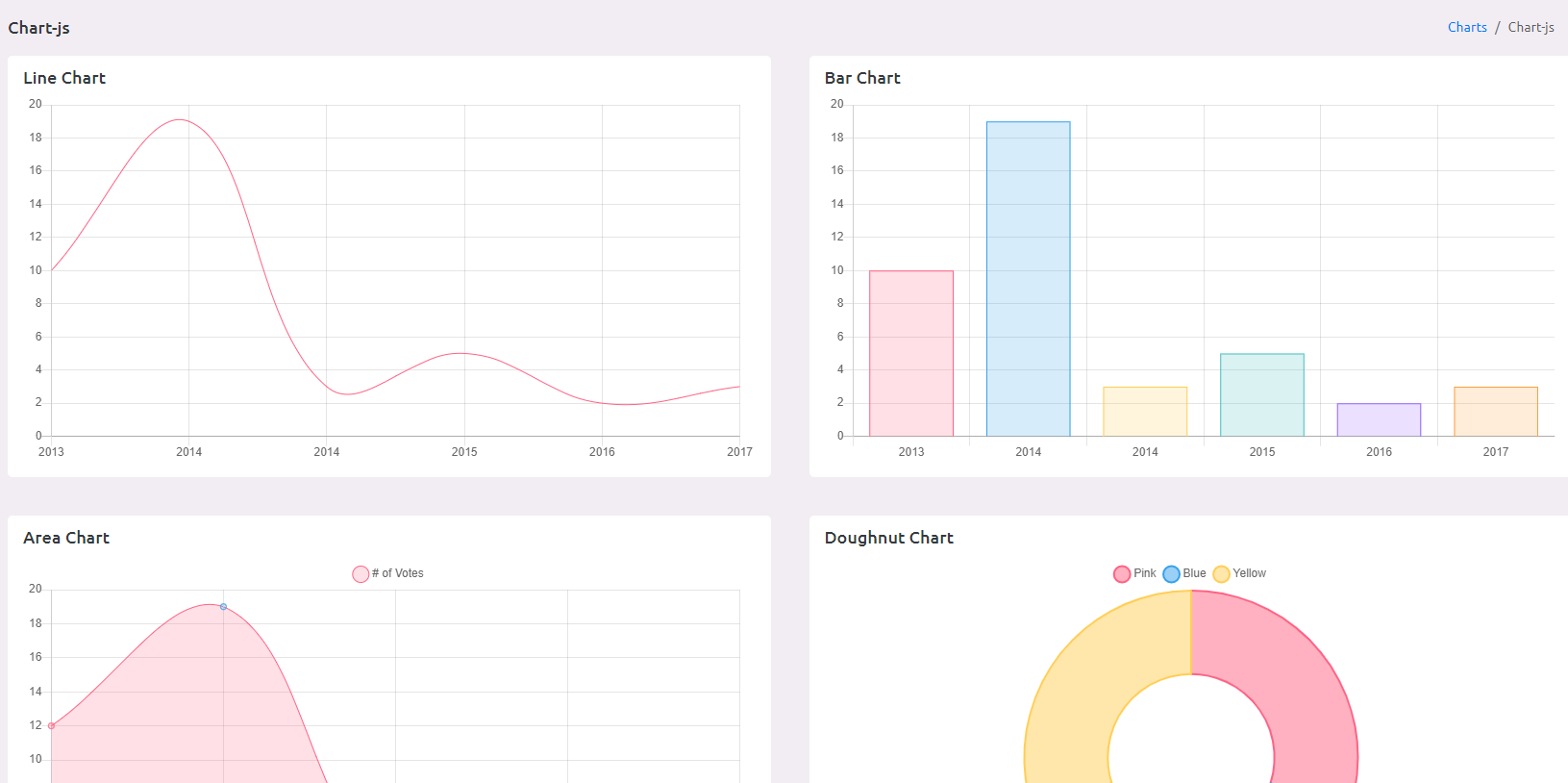


根据接口名称可以查看对应的日志记录。



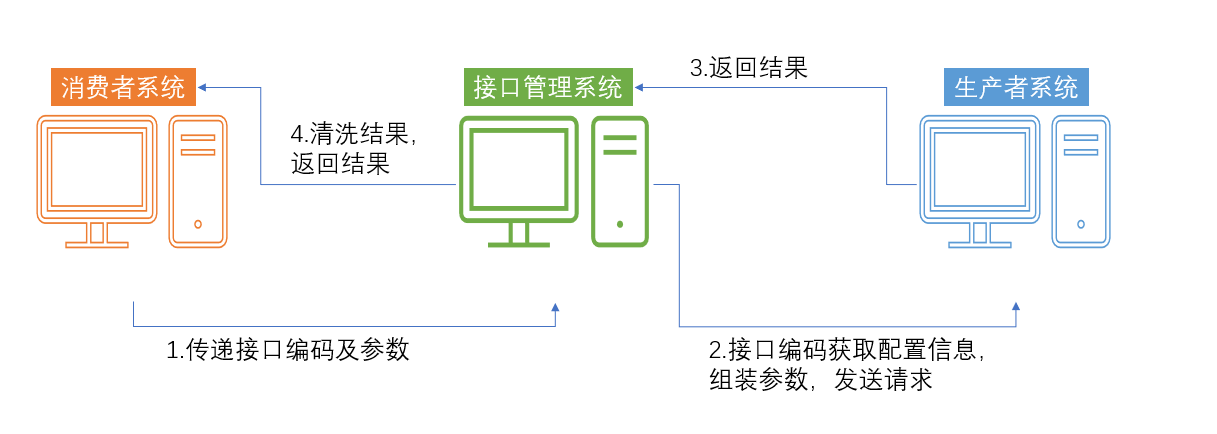
查询信息提供的查询都支持模糊查询。

1. Charts



用于以后扩展，目前只是存放样式。

1. 开发帮助



HTTP

POST

1. 使用SDK生成TOKEN

|  |
| --- |
| CnbmEncryptUtil.setheaderEncrypt(new EncryptBean("demo001")) |

将结果放入请求的header中，key为token

1. 编写接口信息

|  |
| --- |
| @PostMapping("/api/demo3")  public WebserviceResult demo3(@RequestBody String json) throws Exception {  String code = JSON.parseObject(json).getString("code");  String isFlag = JSON.parseObject(json).getString("isFlag");  Map<String, String> resultMap = new HashMap<>();  //对方是map对象时需要使用MultiValueMap才可以  CommonRestFullClient<Result, String> client = new CommonRestFullClient<Result, String>() {  @Override  protected CommonInterfaceResult processResult(Result s) {  return new CommonInterfaceResult(true, JSON.toJSONString(s.getData()));  }  @Override  protected void setCallback(CommonWebservice bean) {  setLog(bean);  }  @Override  protected CommonParam getParams() throws Exception {  CommonParam commonParam = new CommonParam();  commonParam.setIntfCode(code);  commonParam.setCls(Result.class);  commonParam.setParams(isFlag);  commonParam.setHeaderInfo("token", "D8w9B5ni20SAe184uLVan");  commonParam.setMethod(HttpMethod.POST);//可以不用填写默认就是post  commonParam.setMediaType(MediaType.APPLICATION\_FORM\_URLENCODED);  //请求上下文格式，表单还是json  return commonParam;  }  };  boolean isSuccess = client.processInfo();  if (!isSuccess) {  resultMap.put("isFlag", "N");  resultMap.put("error", client.getResultMsg());  }  return WebserviceResult.resultData(resultMap);  } |

HTTP请求通过CommonRestFullClient抽象类，可以实现GET和POST请求，匿名接口中有三个实现方法，

processResult：请求返回结果，返回结果对象根据CommonRestFullClient定义的转换

setCallback：不需要修改，提供给父类的。

|  |
| --- |
| protected void setCallback(CommonWebservice bean) {  setLog(bean);  } |

getParams：请求参数信息，可以设置返回类型，重要参数setIntfCode必须传递。

setHeaderInfo设置头信息参数，对方有权限校验时可以设定。

setMethod接口方法，GET/POST

setMediaType 请求上下文格式，表单还是json

**注：**

**表单传递参数时需要使用**

**MultiValueMap<String, String> valueMap = new LinkedMultiValueMap<>();不然无法进行接收。**

GET

|  |
| --- |
| @PostMapping("/api/demo4")  public WebserviceResult demo4(@RequestBody String json) throws Exception {  String code = JSON.parseObject(json).getString("code");  String id = JSON.parseObject(json).getString("id");  Map<String, String> resultMap = new HashMap<>();  //对方是map对象时需要使用MultiValueMap才可以  CommonRestFullClient<Result, String> client = new CommonRestFullClient<Result, String>() {  @Override  protected CommonInterfaceResult processResult(Result s) {  resultMap.put("data", JSON.toJSONString(s.getData()));  return new CommonInterfaceResult(true, s.getData());  }  @Override  protected void setCallback(CommonWebservice bean) {  setLog(bean);  }  @Override  protected CommonParam getParams() throws Exception {  CommonParam commonParam = new CommonParam();  commonParam.setIntfCode(code);  commonParam.setCls(Result.class);  commonParam.setParams("id=" + id);  //get如果不是自己封装参数，那么就需要使用Map对象方式，系统自动会封装  commonParam.setHeaderInfo("token", "D8w9B5ni20SAe184uLVanv9nZPsn");  commonParam.setMethod(HttpMethod.GET);//可以不用填写默认就是post  commonParam.setMediaType(MediaType.APPLICATION\_FORM\_URLENCODED);//请求上下文格式，表单还是json  return commonParam;  }  };  boolean isSuccess = client.processInfo();  if (!isSuccess) {  resultMap.put("isFlag", "N");  resultMap.put("error", client.getResultMsg());  }  return WebserviceResult.resultData(resultMap);  } |

HTTP请求通过CommonRestFullClient抽象类，可以实现GET和POST请求，匿名接口中有三个实现方法，

processResult：请求返回结果，返回结果对象根据CommonRestFullClient定义的转换

setCallback：不需要修改，提供给父类的。

|  |
| --- |
| protected void setCallback(CommonWebservice bean) {  setLog(bean);  } |

getParams：请求参数信息，可以设置返回类型，重要参数setIntfCode必须传递。

setHeaderInfo设置头信息参数，对方有权限校验时可以设定。

setMethod接口方法，GET/POST

setMediaType 请求上下文格式，表单还是json

GET提供两种方式传递参数：

第一种方式：

<http://localhost:8088/cnbm/intf/get?id=1> commonParam.setParams("name=zhangsan");直接赋值

http://localhost:8088/cnbm/intf/get 在commonParam.setParams("name=zhangsan");直接赋值

第二种方式：

|  |
| --- |
| http://localhost:8088/cnbm/intf/get  Map<String, String> params = new HashMap<>();  params.put("id", "1");  在commonParam.setParams(params);直接赋值时使用Map对象,  http://localhost:8088/cnbm/intf/get?id=1  Map<String, String> params = new HashMap<>();  params.put("id", "1");  在commonParam.setParams(params);直接赋值时使用Map对象, |

**注：**

**表单传递参数时需要使用**

**MultiValueMap<String, String> valueMap = new LinkedMultiValueMap<>();不然无法进行接收。**

WEBSERVICE

字符串

|  |
| --- |
| @PostMapping("/api/demo1")  public WebserviceResult demo1(@RequestBody String json) {  Demo1 demo1 = JSON.parseObject(json, Demo1.class);  String code = JSON.parseObject(json).getString("code");  Map<String, String> resultMap = new HashMap<>();  CommonInterfaceStrClient client = new CommonInterfaceStrClient() {  @Override  protected void setCallback(CommonWebservice bean) {  setLog(bean);  }  @Override  protected CommonParam getParams() throws Exception {  CommonParam commonParam = new CommonParam();  commonParam.setReturnMap(false);  commonParam.setIntfCode(code);  commonParam.setParams(demo1);  commonParam.setCls(Demo1Out.class);  return commonParam;  }  @Override  protected boolean CommonReturnMessage(Map<String, Object> returnMessage) {  return false;  }  @Override  protected boolean CommonReturnMessage(Object returnObject) {  Demo1Out result = (Demo1Out) returnObject;  if (null != result) {  resultMap.put("result", JSON.toJSONString(result));  resultMap.put("isFlag", result.getIsFlag());  return true;  } else {  resultMap.put("error", getResultMsg());  }  return false;  }  };  boolean isSuccess = client.processInfo();  if (!isSuccess) {  resultMap.put("isFlag", "N");  resultMap.put("error", client.getExceptionMsg());  }  return WebserviceResult.resultData(resultMap);  } |

公共信息跟HTTP是一样的，字符串模式使用CommonInterfaceStrClient抽象类来进行调用。

setReturnMap:true/false 返回结果支持map和自定义对象，如果为false时需要设置setCls对象

参数信息：将通过String类型里面存放xlm格式进行传递。

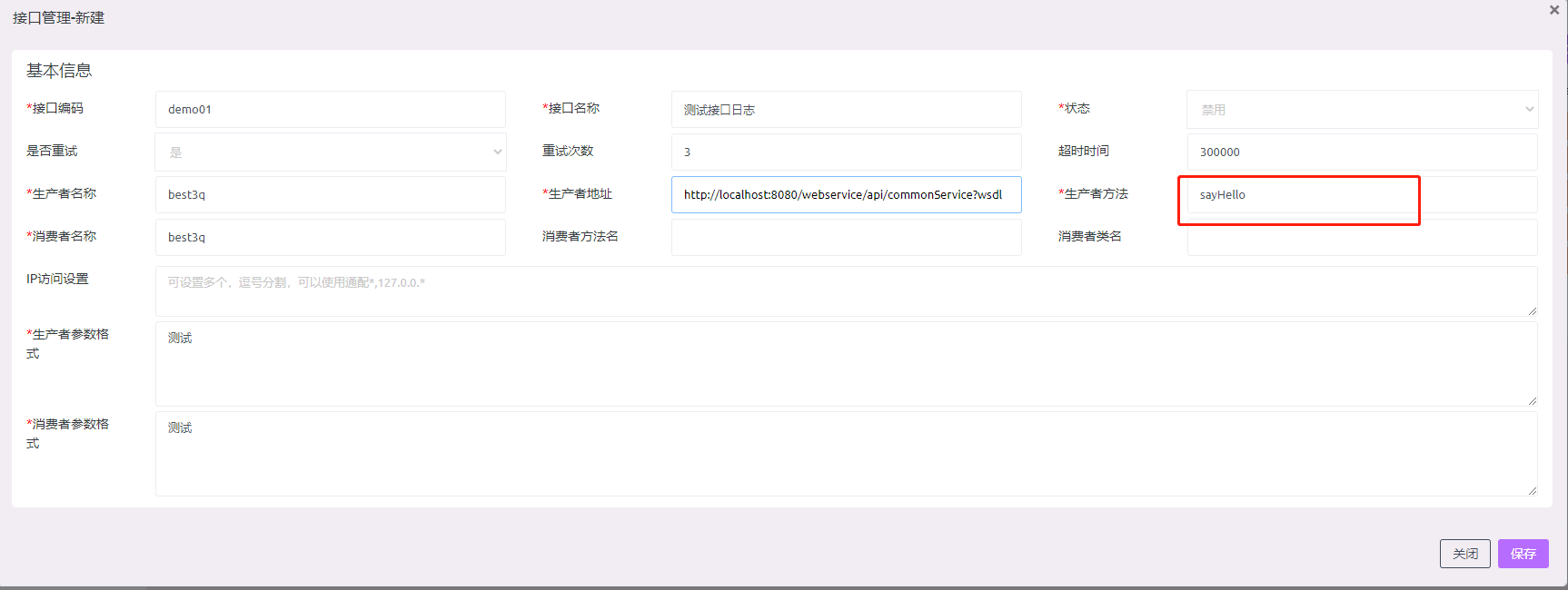
对象

|  |
| --- |
| @PostMapping("/api/demo2")  public WebserviceResult demo2(@RequestBody String json) throws Exception {  String code = JSON.parseObject(json).getString("code");  String name = JSON.parseObject(json).getString("name");  Map<String, String> resultMap = new HashMap<>();  CommonWebServiceClient<CommonServiceStub> client = new CommonWebServiceClient<CommonServiceStub>() {  @Override  protected CommonParam getParams() throws Exception {  CommonParam commonParam = new CommonParam();  commonParam.setIntfCode(code);  CommonServiceStub.SayHelloE sayHello = new CommonServiceStub.SayHelloE();  CommonServiceStub.SayHello hello = new CommonServiceStub.SayHello();  hello.setUserName(name);  sayHello.setSayHello(hello);  commonParam.setParams(sayHello);  commonParam.setCls(CommonServiceStub.class);  return commonParam;  }  @Override  protected CommonInterfaceResult processResult(Object t) {  CommonServiceStub.SayHelloResponseE result = (CommonServiceStub.SayHelloResponseE) t;  System.out.println("返回结果:" + result.getSayHelloResponse().getString());  resultMap.put("isFlag", "Y");  resultMap.put("data", result.getSayHelloResponse().getString());  return new CommonInterfaceResult(true, result.getSayHelloResponse().getString());  }  @Override  protected void setCallback(CommonWebservice bean) {  setLog(bean);  }  };  boolean isSuccess = client.processInfo();  if (!isSuccess) {  resultMap.put("isFlag", "N");  resultMap.put("error", client.getExceptionMsg());  }  //throw new Exception("dsdsd");  return WebserviceResult.resultData(resultMap);  } |

公共信息跟HTTP是一样的，对象模式使用CommonWebServiceClient抽象类来进行调用。

setReturnMap:true/false 返回结果支持map和自定义对象，如果为false时需要设置setCls对象

CommonWebServiceClient设置对象为wsdl生成的类，同时在页面配置接口信息时需要配置调用方法。



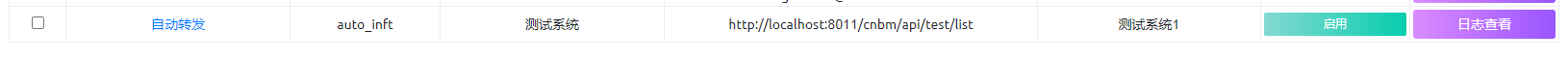
1. 协议转发

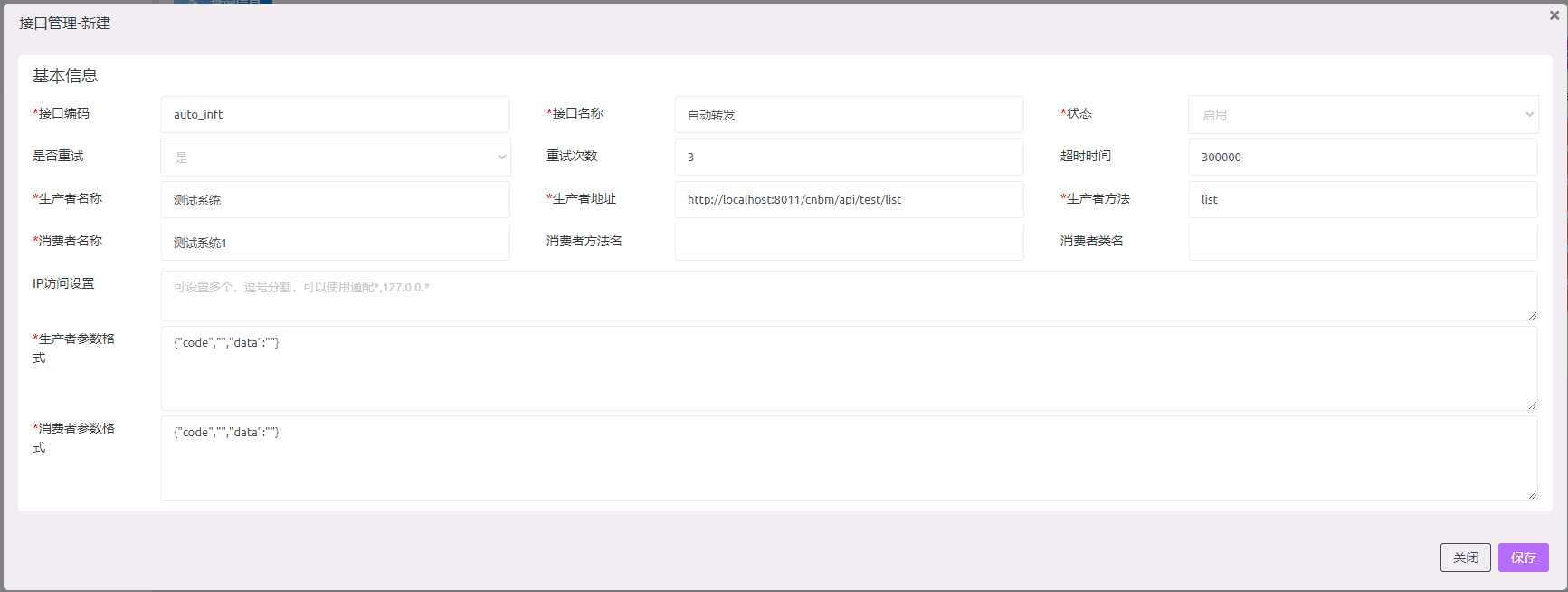
不做数据处理，直接转发接口并记录日志。

接口调用公共地址：/api/common/intf /postman，工程前缀需要完整。

使用方法：

1. 接口管理配置





1. 接口参数格式

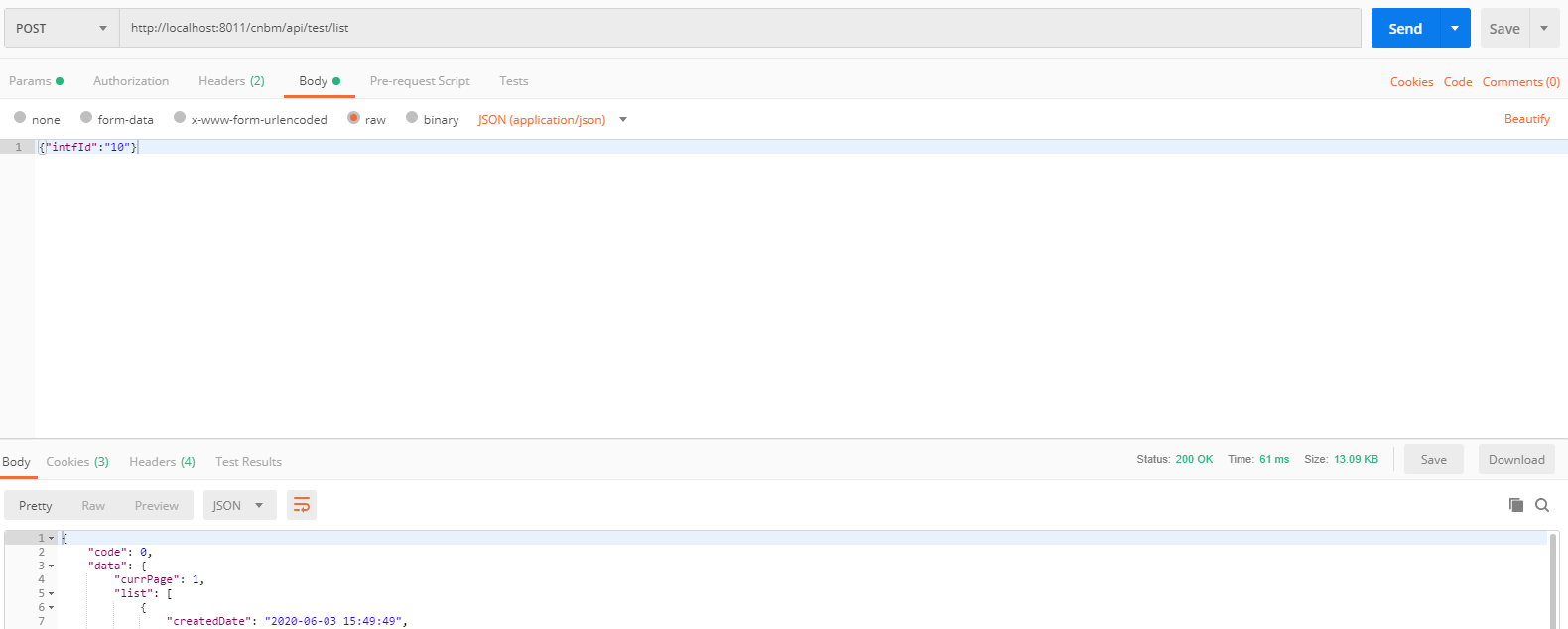
|  |
| --- |
| {  "code":"auto\_inft",  "method": "get",  "data":{  源接口参数信息  }  } |

code: 接口编码

method:接口调用类型，get/post 默认post

data:传递原来接口参数信息

实例：原接口信息



使用接口平台自动转发接口：

