**自动化平台开发**

|  |
| --- |
| 前后端数据结构更改需要进行通告 |

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **编辑人** | **编辑内容** | **备注** |
| 2023/5/8 | 黄兴华 | 自动化平台初稿 |  |
| 2023/5/8 | 张军 | 平台框架图 | v1.0 |
| 2023/5/8 | 张军 | 技术栈选型 | v1.0 |
| 2023/5/8 | 张军 | 后端框架设计 | v1.0 |
| 2023/5/9 | 张军 | 配置服务接口开发 | v1.0 |
| 2023/5/10 | 张军 | 配置服务前端开发 | v1.0 |
| 2023/5/13 | 张军 | 修改用例数据结构 | v1.0 |
| 2023/5/14 | 张军 | 1、修改配置功能 2、显示用例列表 3、前端实现用例多条件组合过滤 | v1.0 |
| 2023/5/15 | 张军 | 文件上传功能 | v1.0 |
| 2023/5/15 | 张军 | 支持批量执行用例 | v1.0 |
| 2023/5/18 | 黄兴华 | 更新测试框架模块版本兼容 | v1.0 |
| 2023/5/18 | 张军 | 1、测试结果报告文件导入样式解析为嵌入式样式 2、下载报告接口 3、定时任务关联用例接口 | v1.0 |
| 2023/5/19 | 张军 | websocket通信建立执行推送 | v1.0 |
| 2023/5/19 | 张军 | 网关转发websocket路由 | v1.0 |
| 2023/5/29 | 黄兴华 | 用例格式更改 | v1.1 |
| 2023/5/30 | 张军 | 下载用例功能 | v1.0 |
| 2023/6/1 | 张军 | 整合缓存中间件 | v1.0 |
| 2023/6/13 | 张军 | 修改springcache缓存更新 | v1.0 |
| 2023/6/15 | 张军 | 整合echarts统计模块 | v1.0 |
| 2023/7/1 | 张军 | 整合luckyexcel支持在线编辑用例 | v1.0 |
| 2023/7/17 | 张军 | 升级nodejs版本、 修改项目启动方式、通过serve服务器启动 | v1.0 |

**平台框架图（2023/5/8）：**

**[框架执行图.pdf]**

**技术栈选型（2023/5/8）：**

1、前端选型：

渐进式框架：vue 2.6.1.0

请求库：axios 0.18.1

前端服务器：node.js 10.16.3

ui框架：element-ui 2.13.2

2、后端选型：

后端语言：java 1.8

服务端框架：spring boot 2.2.1

微服务组件：spring cloud Hoxton.RELEASE

缓存：spring cache

中间件：redis 3.2.100

注册中心：alibaba-nacos 1.4.1

数据库：mysql 5.7

持久层框架：mybatis-plus 3.0.5

接口文档管理：swagger 2.7.0

3、自动化程序：

自动化语言：python：3.9.13

其他模块版本号：

os：10

urllib3：1.26.15

xlrd：2.0.1

json：2.0.9

requests：2.29.0

logging：0.5.1.2

自动化框架模块：

Pytest-html: 3.1.1  
Pytest-xdist: 2.4.0  
Pytest-ordering: 0.6  
Pytest-rerunfailures: 10.2

pytest: 6.2.4

自动化测试报告：

allure-pytest: 2.10.0

指令：

生成allure报告数据  
pytest --alluredir=./report/result

生成allure报告

allure generate ./report/result --clean -o ./report/html

打开allure报告

allure open ./report/html

pytest-html：3.1.1

指令：

生成html指令：

./report=目录

report.html=文件.html

批量执行命令：pytest -vs --html=report.html -k TestExample --capture=sys

**服务拆分（2023/5/8）：**

基础依赖：

接口管理：springfox-swagger2

缓存中间件：spring-boot-starter-data-redis

通信连接：spring-boot-starter-websocket

基础设施服务：

service-gateway：网关服务，做请求转发和跨域处理

访问端口：8000

业务服务：

quyi-user：用户模块，提供临时登录

访问端口：8001

quyi-test-config：全局配置管理模块，管理执行自动化的全局配置

访问端口：8002

quyi-api-autotest：接口自动化管理模块，管理执行所有的自动化用例

访问端口：8003

quyi-test-result：测试结果模块、读取自动化程序返回的结果内容返回给前端渲染

访问端口：8004

quyi-test-task：定时任务模块、提供自动执行测试任务的功能

访问端口：8006

**访问地址（2023/5/8）：**

Mysql 地址：192.168.5.213:3306

账号：root

密码：123456

redis地址：192.168.5.213:6379

swagger接口文档：

访问地址：http://192.168.5.213:${服务端口号}/swagger-ui.html

nacos地址:

访问地址：http://192.168.5.213:8848/nacos

账号：nacos

密码：nacos

前端页面访问：

http://192.168.5.213:3000 admin/123456

后端接口访问地址：

http://192.168.5.213:8000

github地址：

后端项目源码：https://github.com/1427554612/QuYiTest

前端项目源码：https://github.com/1427554612/quyi-autotest-vue

自动化程序源码：https://github.com/gawodeyaoziba/automation\_toold

**数据库设计：**

**配置服务对应的数据库：quyi-test-config**

test\_config表：

DROP TABLE IF EXISTS test\_config;

CREATE TABLE test\_config (

config\_id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '配置表id',

config\_name varchar(20) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '配置名称',

config\_data json NULL COMMENT '配置参数',

config\_type char(10) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '配置类型:取值 web-uiapiphone-uiperformance',

create\_time datetime(0) NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP(0) COMMENT '创建时间',

update\_time datetime(0) NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP(0) COMMENT '修改时间',

update\_up varchar(20) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '修改人',

config\_mark tinytext CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL COMMENT '备注'

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

**结果服务对应的数据库：quyi-test-result**

表关联关系：1 对 多

test\_result：结果表

DROP TABLE IF EXISTS test\_result;

CREATE TABLE test\_result (

result\_id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '结果id',

case\_name varchar(50) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '用例名称',

case\_type varchar(20) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '用例类型',

run\_num int(11) NULL DEFAULT NULL COMMENT '执行次数',

run\_success\_num int(11) NULL DEFAULT NULL COMMENT '执行成功次数',

run\_error\_num int(11) NULL DEFAULT NULL COMMENT '执行失败次数',

run\_success\_rate double NULL DEFAULT NULL COMMENT '执行成功率',

last\_run\_date datetime(0) NULL DEFAULT NULL COMMENT '最近一次执行时间',

last\_run\_time int(11) NULL DEFAULT NULL COMMENT '最近一次执行耗时'

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

test\_result\_info：结果详情表

DROP TABLE IF EXISTS test\_result\_info;

CREATE TABLE test\_result\_info (

result\_info\_id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '详情id',

result\_id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '结果id',

run\_result tinyint(1) NULL DEFAULT NULL COMMENT '是否成功,0表示错误、1表示成功',

run\_begin\_time datetime(0) NULL DEFAULT NULL COMMENT '开始执行时间',

run\_end\_time datetime(0) NULL DEFAULT NULL COMMENT '执行结束时间',

run\_time int(11) NULL DEFAULT NULL COMMENT '执行时长'

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

**定时任务服务对应的数据库：quyi-test-task**

表关联关系：1 对 多

test\_task：任务表

DROP TABLE IF EXISTS test\_task;

CREATE TABLE test\_task(

id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '任务id',

name varchar(30) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '任务名称',

type char(4) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '执行类型',

date int(10) NULL DEFAULT NULL COMMENT '执行毫秒数',

cron varchar(30) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT 'cron表达式',

cron\_parse varchar(30) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT 'cron表达式的意思',

jobs json NULL COMMENT '执行所需数据',

create\_time datetime(0) NULL DEFAULT NULL COMMENT '创建时间',

update\_time datetime(0) NULL DEFAULT NULL COMMENT '修改时间',

create\_by varchar(20) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '创建人'

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

task\_run\_log：任务执行日志表：

DROP TABLE IF EXISTS task\_run\_log;

CREATE TABLE task\_run\_log (

id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '执行id',

begin\_time datetime(0) NULL DEFAULT NULL COMMENT '开始执行时间',

end\_time datetime(0) NULL DEFAULT NULL COMMENT '执行结束时间',

success\_count int(5) NULL DEFAULT NULL COMMENT '正确执行的用例条数',

error\_count int(5) NULL DEFAULT NULL COMMENT '执行失败的用例总数',

task\_id char(19) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '任务id',

run\_status int(1) NULL DEFAULT NULL COMMENT '运行状态、0表示未开始、1表示执行中、2表示已终止'

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

**Redis缓存说明：**

**db0：存储test-config服务数据**

ConfigPath::configPath：

key说明：初始化配置数据

TTL：一天

TestConfig::{configId}:

key说明：具体配置数据

TTL：一小时

TestConfig::list：

key说明：所有配置信息

TTL：一小时

TestConfig::page\_{current}\_{size}:

key说明：分页查询的配置信息

TTL：一小时

**db1：存储api-autotest服务数据**

TestCase::list：

key说明：所有测试用例列表

TTL：一天

**db2：存储test-result服务数据**

**db3：存储test-task服务数据**

TestTask::{taskId}

key说明：定时任务信息

TTL：一小时

TestTask::list

key说明：所有定时任务信息

TTL：一天

TestTask::page\_{current}\_{size}

key说明：分页查询的定时任务信息

TTL：一小时

RelationCase::{taskId}

key说明：已关联用例的任务信息

TTL：一小时

**他是如何工作的？**

**第一步：定义excel格式（2023/5/8）**

1、用例编号：

对应实体字段：caseNumber

含义：用例的唯一标识

是否必填：是

2、用例名称：

对应实体字段：caseName

含义：用例的名称、后端通过该字段生成命令调用py自动化程序

是否必填：是

3、用例标题：

对应实体字段：caseTitle

含义：用例的标题

是否必填：是

4、接口地址：

对应实体字段：apiPath

含义：接口地址、通过该地址请求平台具体接口

是否必填：是

5、请求方式：

对应实体字段：requestMethod

含义：请求的方式、取值：get、post、delete、put 字母规范都为小写

是否必填：是

6、主流程api：

对应实体字段：isMainProcessApi

含义：取值为：是\否、当该值为是的时候、当前用例如果出现异常或断言错误、则停止程序执行

是否必填：是

7、请求头：

对应实体字段：requestHeaders

含义：接口的请求头参数

是否必填：是

数据结构：

{

"User-Agent:": "string",

"authorization": "string"

}

8、请求体：

对应实体字段：requestBody

含义：接口请求体

是否必填：否

数据结构：

{

"account":"#{string}",

"password":"string",

"person":"string",

"grecaptcha\_token":"string"

}

9、是否执行：

对应实体字段：isRun

含义：取值方式、是/否、否表示过滤掉该用例

是否必填：是

10、断言内容：

对应实体字段：assertMap

含义：判断接口执行结果是否正确

是否必填：是

数据结构：

{

"assertType": "and",

"assertList": [{

"key": "code",

"value": 200,

"type": "eq"

},

{

"key": "id",

"value": "1523652388952",

"type": "eq"

}

]

}

11、是否参数化：

对应实体字段：isParams

含义：判断接口是否需要参数化、取值：是/否

是否必填：是

12、参数化数据：

对应实体字段：paramList

含义：具体参数化的信息

是否必填：是

数据结构：

[{

"paramName": "username",

"paramFrom": "requestBody",

"paramsEq": "data.username"

}, {

"paramName": "password",

"paramFrom": "requestBody",

"paramsEq": "data.username"

}]

**excel用例数据结构对应java实体：**

|  |
| --- |
| Java @Data @ApiModel(value = "测试用例对象") @EqualsAndHashCode @ToString public class ApiTestCaseEntity {   @ApiModelProperty(value = "用例编号")  @Excel(name = "用例编号")  private Integer caseNumber;   @ApiModelProperty(value = "用例名称")  @Excel(name = "用例名称")  private String caseName;   @ApiModelProperty(value = "用例标题")  @Excel(name = "用例标题")  private String caseTitle;   @ApiModelProperty(value = "接口地址")  @Excel(name = "接口地址")  private String apiPath;   @ApiModelProperty(value = "请求方式")  @Excel(name = "请求方式")  private String requestMethod;   @ApiModelProperty(value = "主流程api")  @Excel(name = "主流程api")  private String isMainProcessApi;   @ApiModelProperty(value = "请求头")  @Excel(name = "请求头")  private Object requestHeaders;   @ApiModelProperty(value = "请求体")  @Excel(name = "请求体")  private Object requestBody;   @ApiModelProperty(value = "是否执行")  @Excel(name = "是否执行")  private String isRun;   @ApiModelProperty(value = "断言内容")  @Excel(name = "断言内容")  private Object assertMap;   @ApiModelProperty(value = "是否参数化")  @Excel(name = "是否参数化")  private String isParams;   @ApiModelProperty(value = "参数化数据")  @Excel(name = "参数化数据")  private Object paramList; } |

**excel格式展示：**

**[api-auto-test.xls]**

**第二步：定义自动化所需配置**

**数据格式展示：config.json**

{

"current": "PHP",

"adminUrl": "xxxx",

"clientUrl": "xxxx",

"excelPath": "d:/api-auto-test.xls",

"threadNum": 1

}

**字段说明：**

current：当前币种 -->String

adminUrl：后台接口地址 -->String

clientUrl：前台接口地址 -->String

excelPath：excel文件的存放路径 -->String

threadNum：线程数-->int

**怎么获得：**

GET http://192.168.5.213:8080/api/[api/test\_config/selectConfigPath](http://localhost:8002/swagger-ui.html#!/test45config45controller/getConfigPathUsingGET)

response:

{

"message": "成功",

"code": 20000,

"data": {

"pythonProjectPath": "D:/quyi-auto-test-project/automation\_toold/exel\_automation",

"logPath": "d:/log",

"reportPath": "d:/report",

"configPath": "d:/config.json",

"apiRunTimePath": "d:/api-runtime.json"

},

"success": true

}

接口调用说明：

自动化程序的初始化配置、在自动化程序初始化的时候调用该接口。得到要读取的用例位置、平台地址和币种等配置。

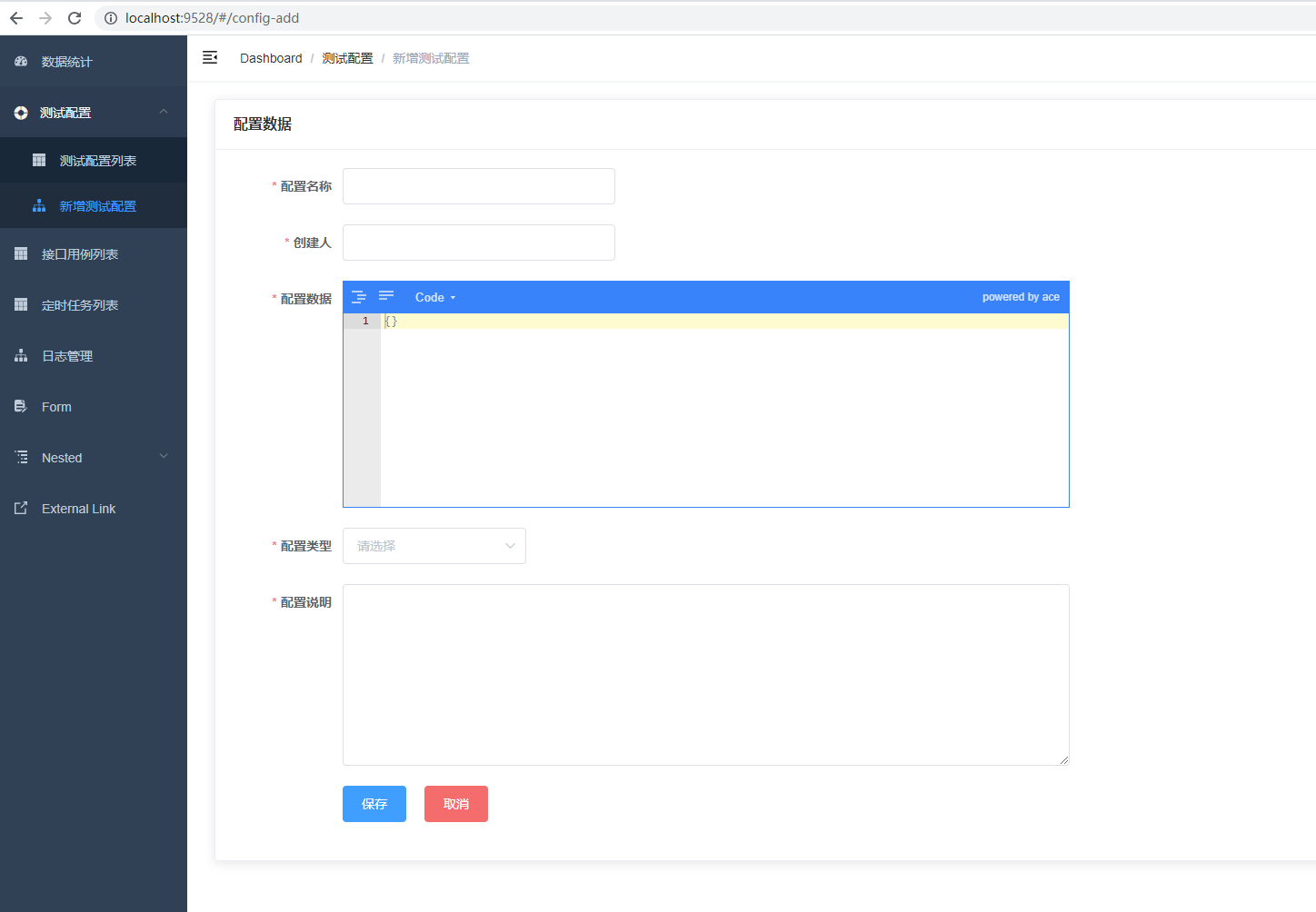
响应返回的配置是动态的还是静态的？

动态的、这些值存放的是nacos配置中心中、每次修改后、再次调用接口、返回的就是修改后的值、无须重新打包部署。

config.json执行的逻辑?

每次前端调用后、通过用例所关联的配置、去查询数据库中的配置表中的config数据、将该数据更新到config.json文件中、使得每次执行都可以选择最新的配置去跑用例

**第三步：进入前端页面添加配置信息**



**对应接口**

[POST](http://localhost:8002/swagger-ui.html#!/test45config45controller/saveTestConfigUsingPOST) http://192.168.5.213:8000[/api/test\_config/saveTestConfig](http://localhost:8002/swagger-ui.html#!/test45config45controller/saveTestConfigUsingPOST)

请求体：

{

"configData": {},

"configId": "string",

"configMark": "string",

"configName": "string",

"configType": "string",

"createTime": "2023-06-02T03:53:40.489Z",

"updateTime": "2023-06-02T03:53:40.489Z",

"updateUp": "string"

}

是否必填：createTime、updateTime之外的所有参数必填

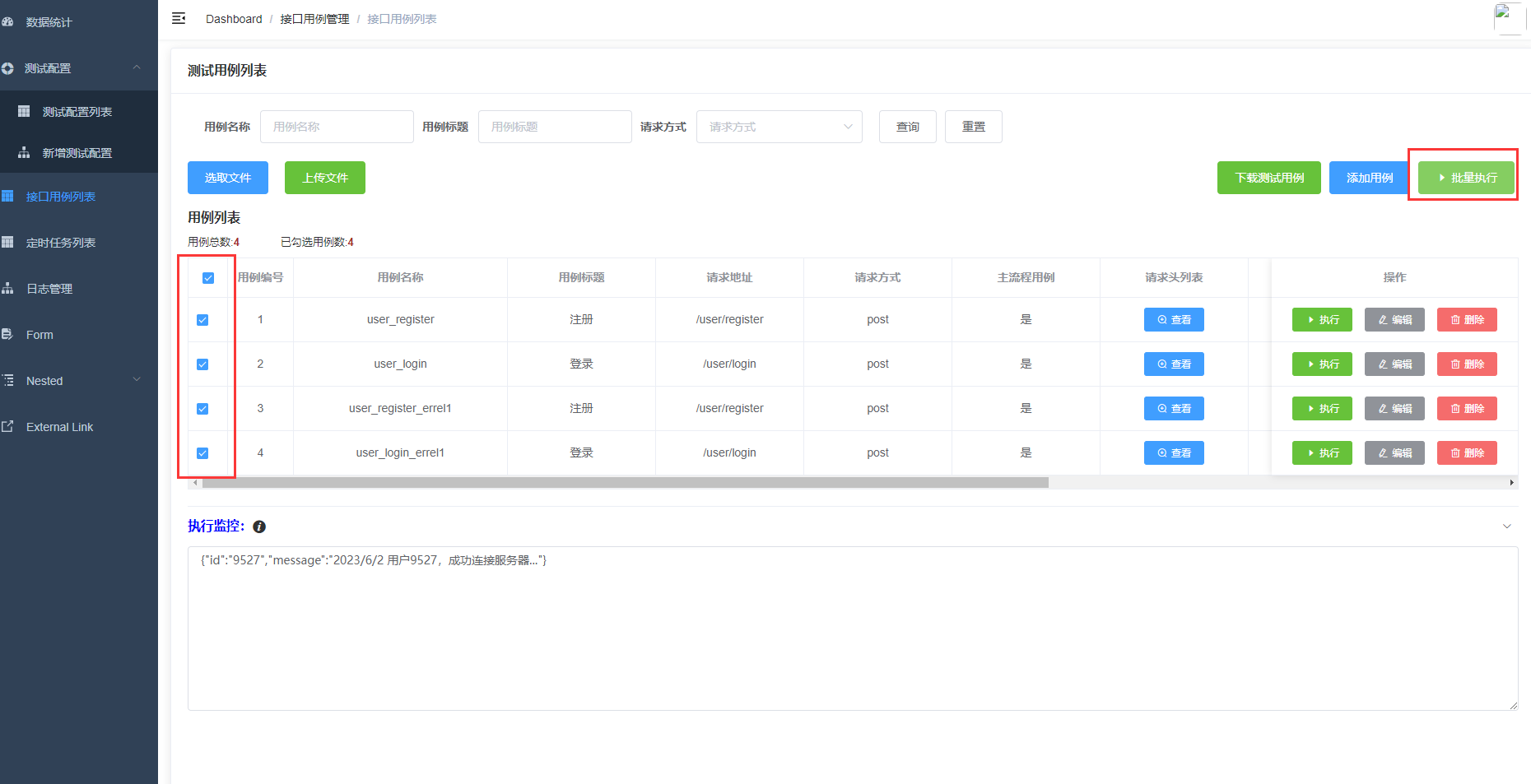
响应：

{  
 "message": "ok",  
 "code": 20000,  
 "data": {},  
 "success": true  
}

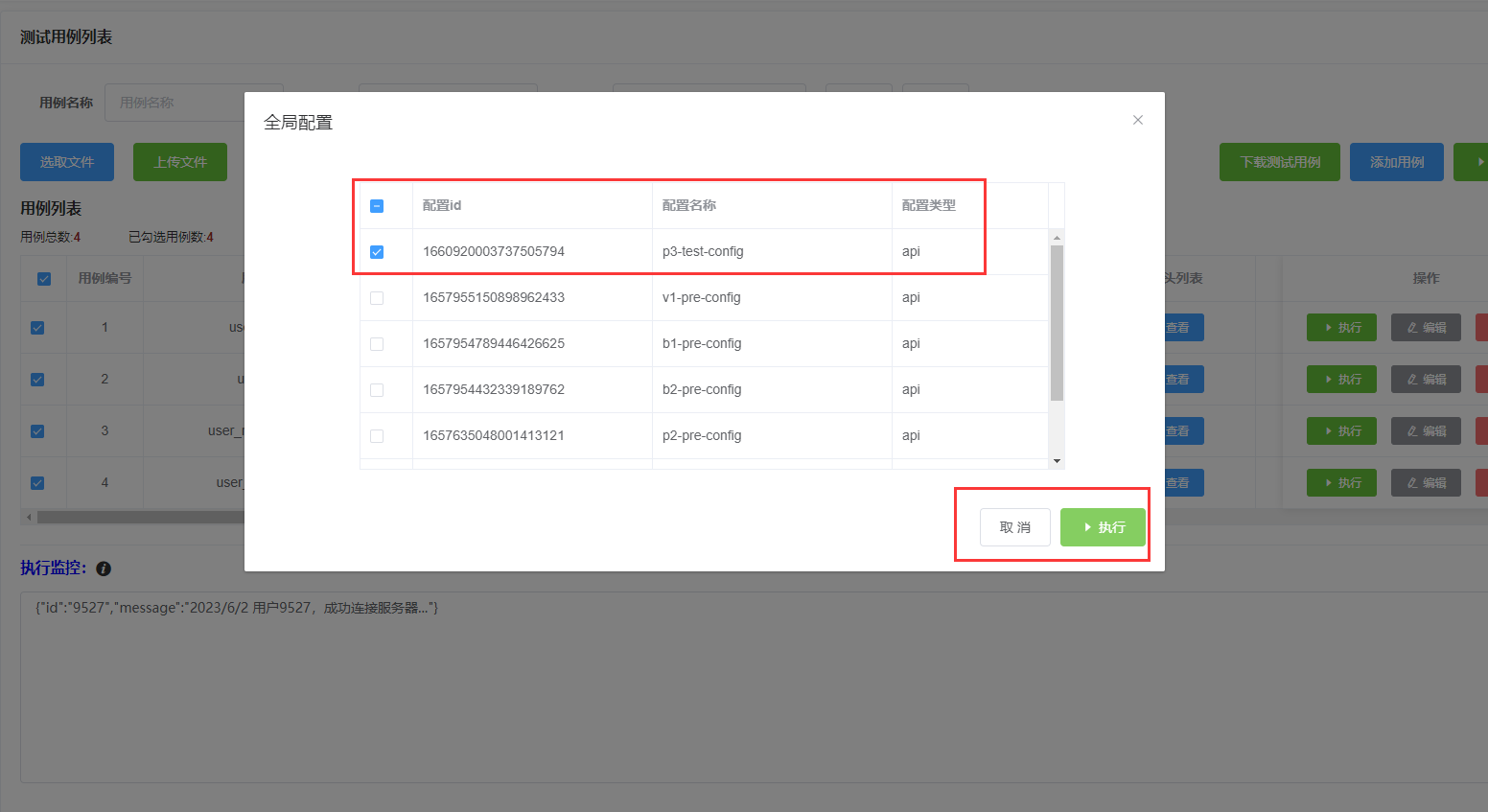
**配置对应java实体：**

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
@ApiModel(value="TestConfig对象", description="")  
// 以resultMap方式返回  
@TableName(autoResultMap = true)  
public class TestConfig implements Serializable {  
  
 private static final long *serialVersionUID* = 1L;  
  
 @ApiModelProperty(value = "配置表id")  
 @TableId(value = "config\_id", type = IdType.*ID\_WORKER\_STR*)  
 private String configId;  
  
 @ApiModelProperty(value = "配置名称")  
 @NotNull(message = "配置名称不能为空")  
 @Length(min = 3,max = 20,message = "字符串长度在3到20之间")  
 private String configName;  
  
 @ApiModelProperty(value = "配置参数")  
 // 将前端传递的json数据转化为map传递  
 @TableField(typeHandler = JacksonTypeHandler.class)  
 @NotNull(message = "配置参数不能为空")  
 private Map<String,Object> configData;  
  
 @ApiModelProperty(value = "配置类型:取值 web-ui,api,phone-ui,performance")  
 @NotNull(message = "配置类型不能为空")  
 private String configType;  
  
 @ApiModelProperty(value = "创建时间")  
 private Date createTime;  
  
 @ApiModelProperty(value = "修改时间")  
 private Date updateTime;  
  
 @ApiModelProperty(value = "修改人")  
 private String updateUp;  
  
 @ApiModelProperty(value = "备注")  
 @NotNull(message = "配置备注不能为空")  
 private String configMark;

**第四步：勾选索要执行的测试用例、点击批量执行**



**第五步：关联所要使用的配置、点击执行**



**执行接口：**

POST http://192.168.5.213:8000[/api/api-autotest/runCase/{configId}](http://localhost:8003/swagger-ui.html#!/api45auto45test45controller/runCaseUsingPOST)

请求参数：["user\_register","user\_login"]

响应：

{

message: "成功",

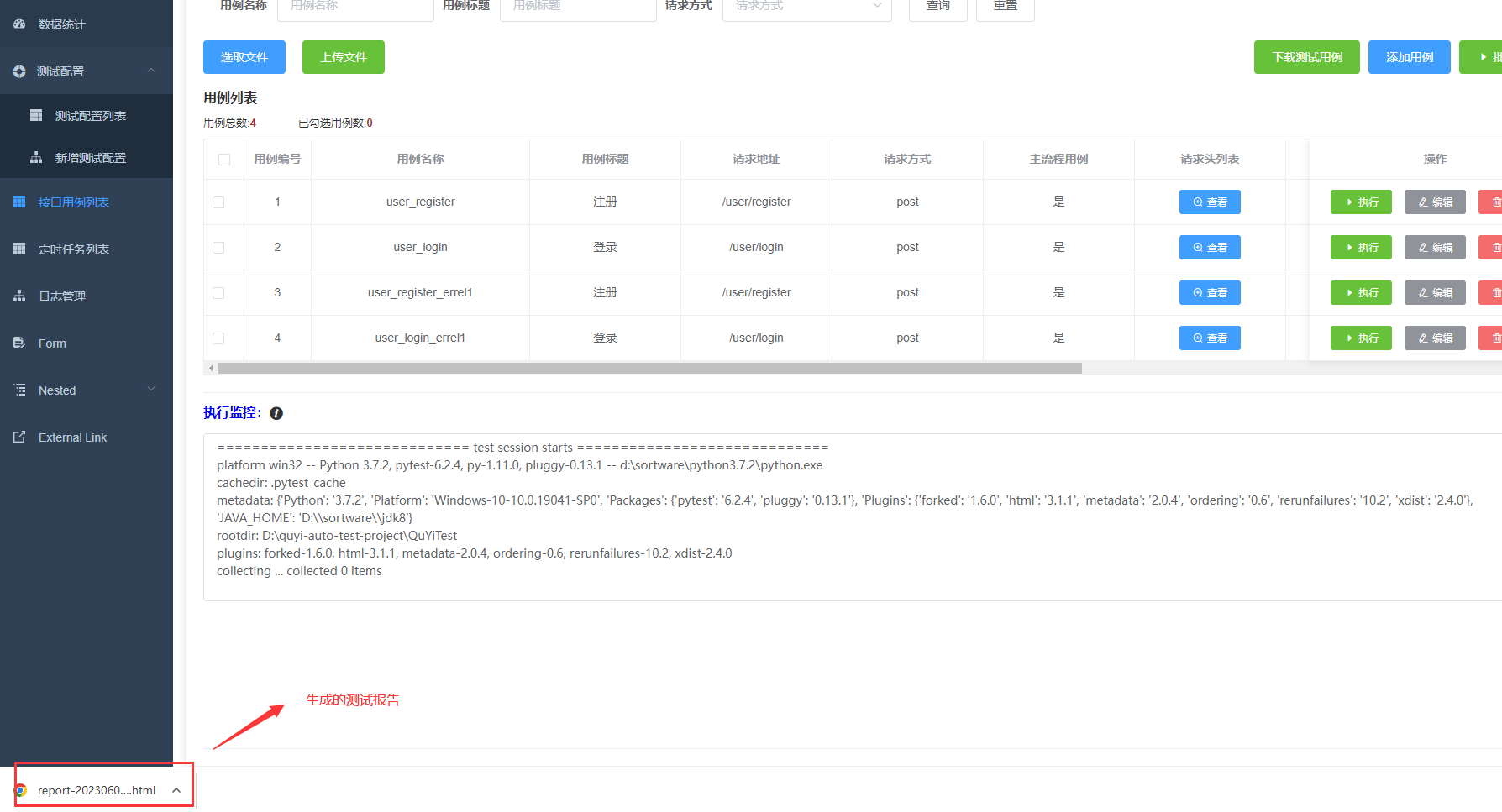
code: 20000,

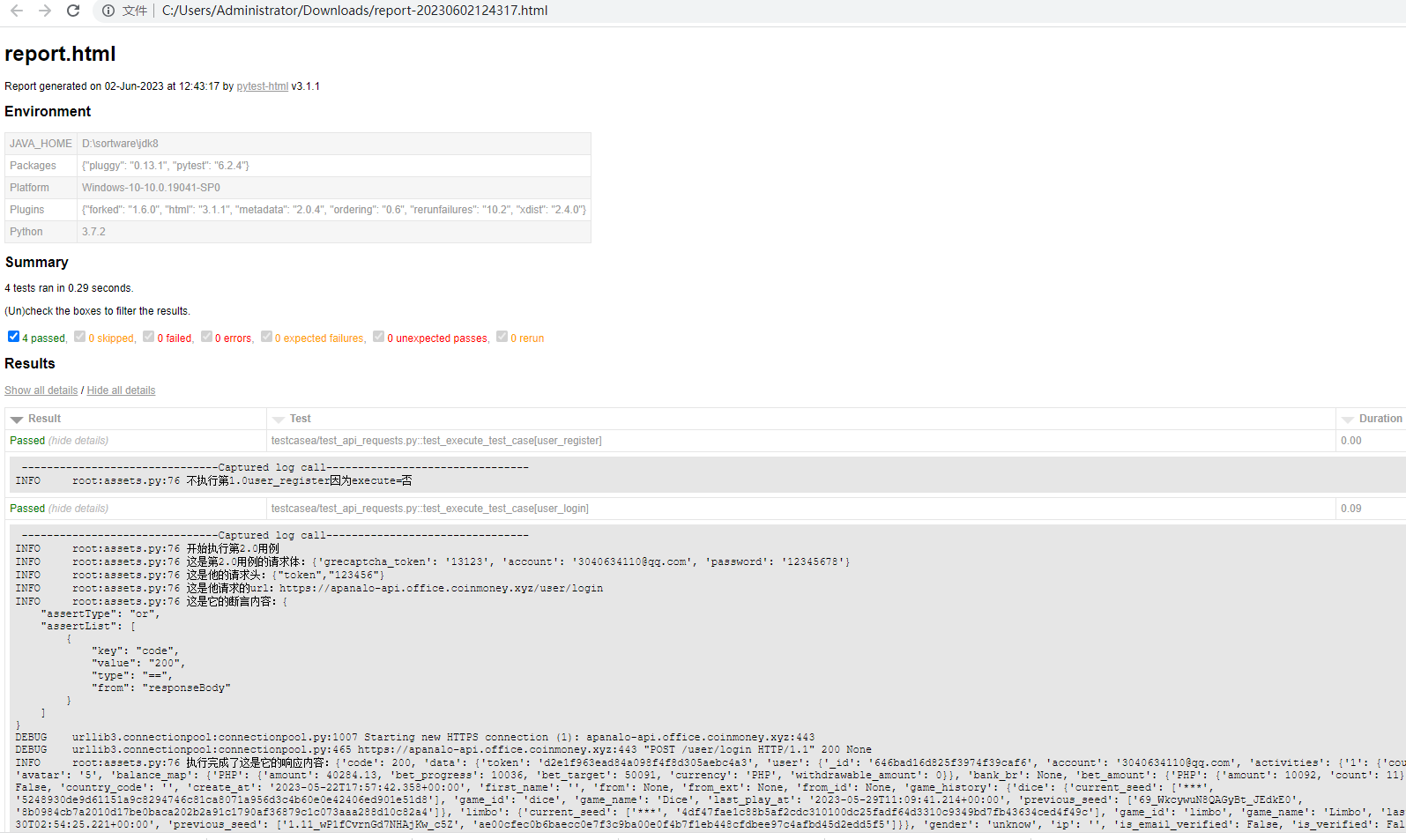
data: {},

success: true

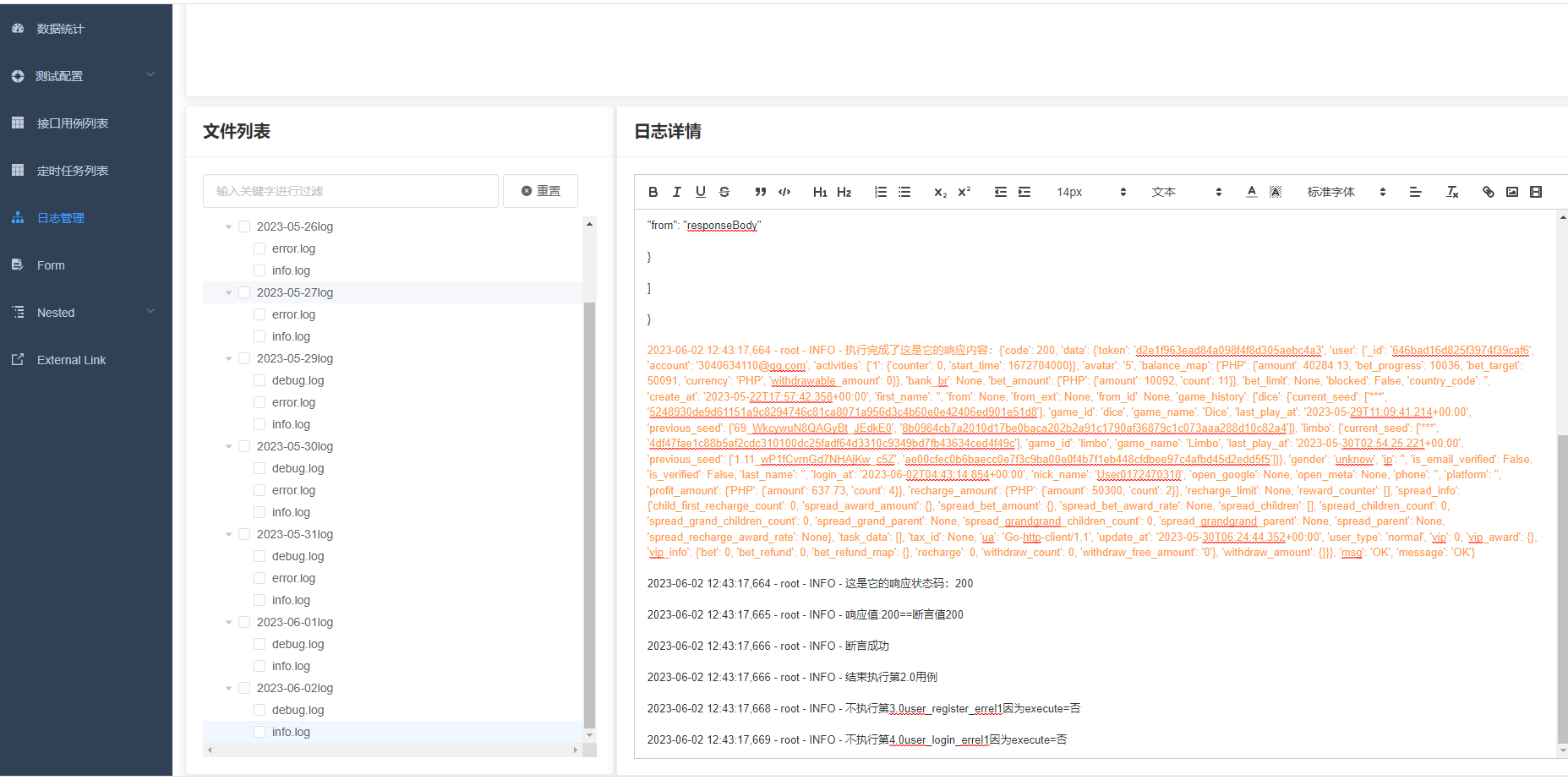
}

**第六步：执行后、自动下载测试报告、打开html文档查看测试结果**





**第七步：查看运行日志**



**日志相关接口：**

查询日志文件树：GET http://192.168.5.213:8000[/api/test\_result/findLogTree](http://localhost:8004/swagger-ui.html#!/test45result45controller/findLogTreeUsingGET)

响应：

{

"message": "成功",

"code": 20000,

"data": {

"list": [

{

"id": "36bcc6c4-d3b7-464c-88d1-2cae69e049f2",

"label": "log",

"path": "d:\\log",

"children": [

{

"id": "d566b4cf-c2ff-4737-8de6-8de780757304",

"label": "2023-05-25log",

"path": "d:\\log\\2023-05-25log",

"children": [

{

"id": "35a92089-6489-4254-a91e-799ffc59184d",

"label": "error.log",

"path": "d:\\log\\2023-05-25log\\error.log",

"children": null,

"file": true

},

{

"id": "3c9f4baf-ed79-44d2-987e-70ccb8a43ff3",

"label": "info.log",

"path": "d:\\log\\2023-05-25log\\info.log",

"children": null,

"file": true

}

],

"file": false

}

添加结果接口：POST <http://192.168.5.213:8000>[/api/test\_result/saveResult](http://localhost:8004/swagger-ui.html#!/test45result45controller/saveResultUsingPOST)

请求参数：

{

"case\_name": "string",

"case\_type": "string",

"last\_run\_date": "2023-06-02T05:53:12.735Z",

"last\_run\_time": 0,

"result\_id": "string",

"run\_error\_num": 0,

"run\_num": 0,

"run\_success\_num": 0,

"run\_success\_rate": 0,

"testResultInfoList": [

{

"result\_id": "string",

"result\_info\_id": "string",

"run\_begin\_time": "string",

"run\_end\_time": "2023-06-02T05:53:12.735Z",

"run\_result": false,

"run\_time": 0

}

]

}

修改结果接口 ：PUT <http://192.168.5.213:8000>[/api/test\_result/updateResult](http://localhost:8004/swagger-ui.html#!/test45result45controller/saveResultUsingPOST)

请求参数：

{

"case\_name": "string",

"case\_type": "string",

"last\_run\_date": "2023-06-02T05:53:12.738Z",

"last\_run\_time": 0,

"result\_id": "string",

"run\_error\_num": 0,

"run\_num": 0,

"run\_success\_num": 0,

"run\_success\_rate": 0,

"testResultInfoList": [

{

"result\_id": "string",

"result\_info\_id": "string",

"run\_begin\_time": "string",

"run\_end\_time": "2023-06-02T05:53:12.738Z",

"run\_result": false,

"run\_time": 0

}

]

}

**对应实体：**

关联关系：一对多

日志结果树对象：

@Data  
public class DataTree {  
 private String id;  
 private String label;  
 private boolean isFile;  
 private String path;  
 private List<DataTree> children;  
}

日志对象：

@TableName(autoResultMap=true)  
@Data  
@EqualsAndHashCode  
@ToString  
@ApiModel(value = "测试结果")  
public class TestResult {  
  
 @TableId(type = IdType.*ID\_WORKER\_STR*)  
 @ApiModelProperty("结果id")  
 private String result\_id;  
  
 @ApiModelProperty("用例名称")  
 private String case\_name;  
  
 @ApiModelProperty("用例类型")  
 private String case\_type;  
  
 @ApiModelProperty("执行次数")  
 private int run\_num;  
  
 @ApiModelProperty("执行成功次数")  
 private int run\_success\_num;  
  
 @ApiModelProperty("执行失败次数")  
 private int run\_error\_num;  
  
 @ApiModelProperty("执行成功率")  
 private double run\_success\_rate;  
  
 @ApiModelProperty("最后一次执行时间")  
 private Date last\_run\_date;  
  
 @ApiModelProperty("最后一次执行耗时")  
 private int last\_run\_time;  
}

日志详情对象：

@Data  
@EqualsAndHashCode  
@ToString  
@ApiModel("结果详情")  
@TableName(autoResultMap = true)  
public class TestResultInfo {  
  
 @ApiModelProperty("详情id")  
 @TableId(type = IdType.*ID\_WORKER\_STR*)  
 private String result\_info\_id;  
  
 @ApiModelProperty("结果id")  
 private String result\_id;  
  
 @ApiModelProperty("执行成功率")  
 private boolean run\_result;  
  
 @ApiModelProperty("执行开始时间")  
 @JsonFormat(shape = JsonFormat.Shape.*STRING*, pattern="yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")  
 private Date run\_begin\_time;  
  
 @ApiModelProperty("执行结束时间")  
 private Date run\_end\_time;  
  
 @ApiModelProperty("执行成功率")  
 private int run\_time;  
}