**CMDB说明文档**

1. 环境说明

|  |  |  |
| --- | --- | --- |
| **名称** | **访问地址** | 说明 |
| CMDB | http://10.1.100.45:8000/user/login | cmdb页面地址 |
| cmdb-api | http://10.1.100.45:8080 | 用于钉钉调用接口更新数据 |
| 手册地址 | https://veops.cn/docs/ | 维易CMDB使用手册 |

1. 模型参数信息
   1. 物理机[模型名称-server]
      1. 字段说明
         1. 必填字段

| **字段名** | **含义** | **示例** | **说明** |
| --- | --- | --- | --- |
| sn | **序列号** | "XSCZL\_GPU4090\_30" | 全局唯一，必填 |
| server\_name | **显示名** | "XSCZL\_GPU4090\_30" | 易读名，必填 |

* + - 1. 常用可选字段

| **字段名** | **含义** | **示例** | **说明** |
| --- | --- | --- | --- |
| private\_ip | ****内网 IP 列表**** | ["10.1.100.11"] | 可填多个 |
| os\_version | ****操作系统**** | "CentOS 7.9" | 如 Ubuntu 20.04 |
| cpu\_count | ****物理 CPU 颗数**** | 2 | 整数 |
| logic\_cpu\_count | ****逻辑 CPU 核数**** | 64 | 整数 |
| ram\_size | ****内存大小**** | "128GB" | 字符串 |
| device\_spec | ****服务器型号**** | "DELL R730" | 字符串 |
| manufacturer | ****厂商**** | "DELL" | 字符串 |
| status | ****状态**** | "在线" | 在线 / 待用 / 下线 / 维修 |
| idc | ****数据中心**** | "张江机房" | 下拉可选值 |
| server\_room | ****机房**** | "2A" | 下拉可选值 |
| rack | ****机架位置**** | "A12-03" | 下拉可选值 |
| u\_count | ****占 U 数**** | 4 | 整数 |
| u\_start | ****起始 U 位**** | 19 | 整数 |
| buy\_date | ****采购日期**** | "2024-07-01" | 日期字符串 |

* + - 1. 网络/管理字段

| **字段名** | **含义** | **示例** | **说明** |
| --- | --- | --- | --- |
| ilo\_ip | ****iLO/BMC IP**** | "10.1.13.30" | 带外管理 |
| ilo\_mac | ****iLO MAC**** | "00:1A:2B:3C:4D:5E" | 字符串 |
| ssh\_port | ****SSH 端口**** | 22 | 整数 |
| vnc\_port | ****VNC 端口**** | 5900 | 整数 |

* + 1. 完整payload示例

{

"ci\_type": "server",

"sn": "GPU\_SERVER\_30",

"server\_name": "GPU服务器\_30",

"private\_ip": ["10.1.100.30"],

"os\_version": "Ubuntu 20.04",

"cpu\_count": 2,

"logic\_cpu\_count": 64,

"ram\_size": "128GB",

"device\_spec": "DELL R730",

"manufacturer": "DELL",

"status": "在线",

"idc": "新疆新丝绸之路数据中心",

"server\_room": "2A",

"rack": "A12-03",

"u\_count": 4,

"u\_start": 19,

"buy\_date": "2024-07-01",

"ilo\_ip": "10.1.13.30",

"ssh\_port": 22

}

curl "http://10.1.100.45:8000/api/v0.1/ci/s?q=\_type:server&\_key=6783474dad5842589811b66492b3d2ae&\_secret=a402d286314ad06dc9c3fbbaabca21c3a716fcee" -H "Content-Type: application/json"

#通过api查询所有server模型的对象信息

* 1. 应用[模型名称-project]
  2. 交换机[模型名称-switch]
  3. 防火墙[模型名称-firewall]
  4. 等等

1. CMDB-API

官方有提供CI、CR操控相关的API接口和python SDK。要实现钉钉流程触发CMDB中服务器的属性和关联关系的更新，现有的API无法解决问题。需要进行二次开发和封装，做一个新的API。

* 1. 功能设计
     1. 更新服务器属性和应用关联



将钉钉表单中的字段数值通过钉钉机器人触发API更新到CMDB相关服务器

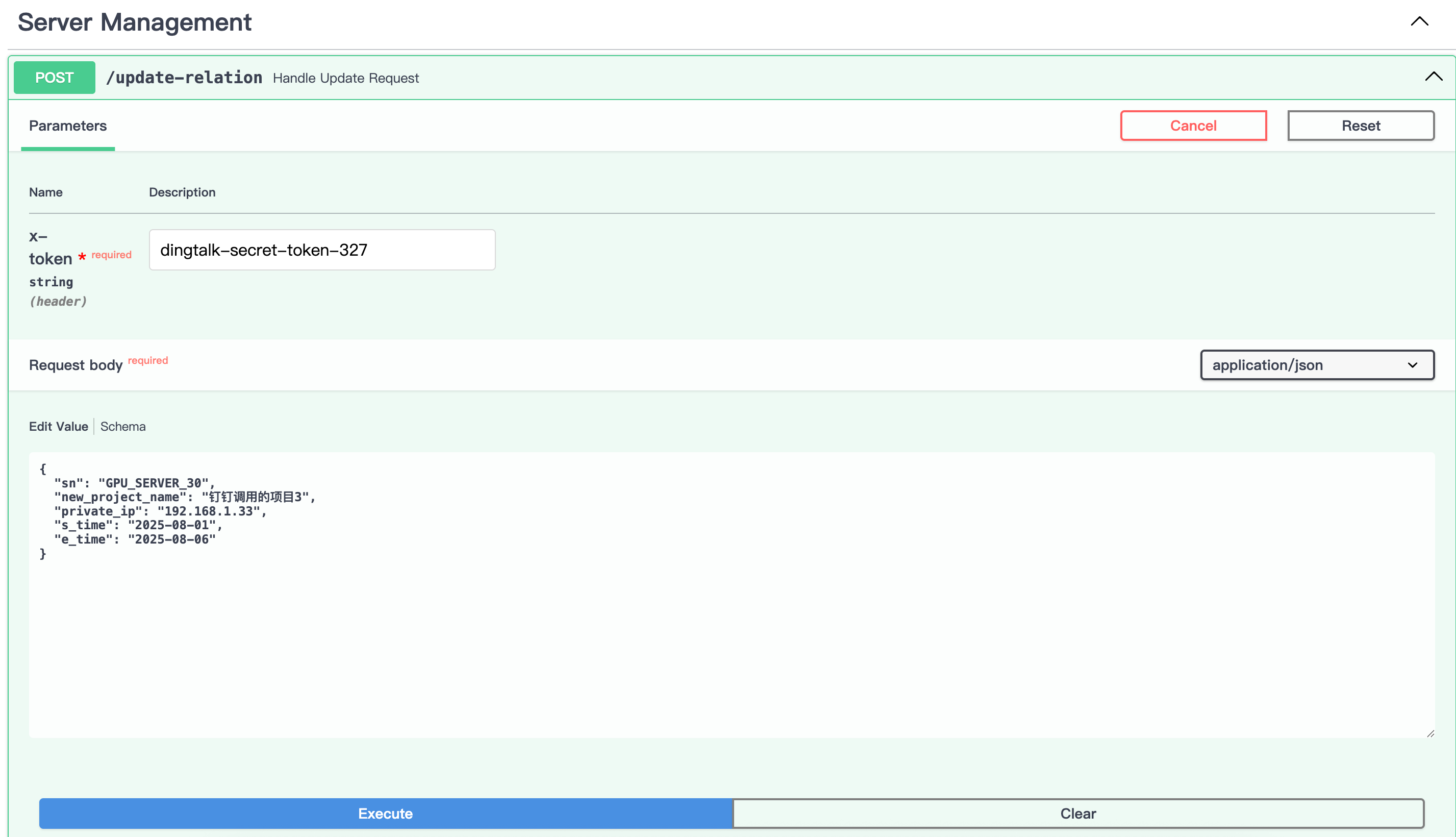
* + 1. 回收服务器属性和应用关联

在钉钉流程进入回收阶段时候触发，清除服务器信息

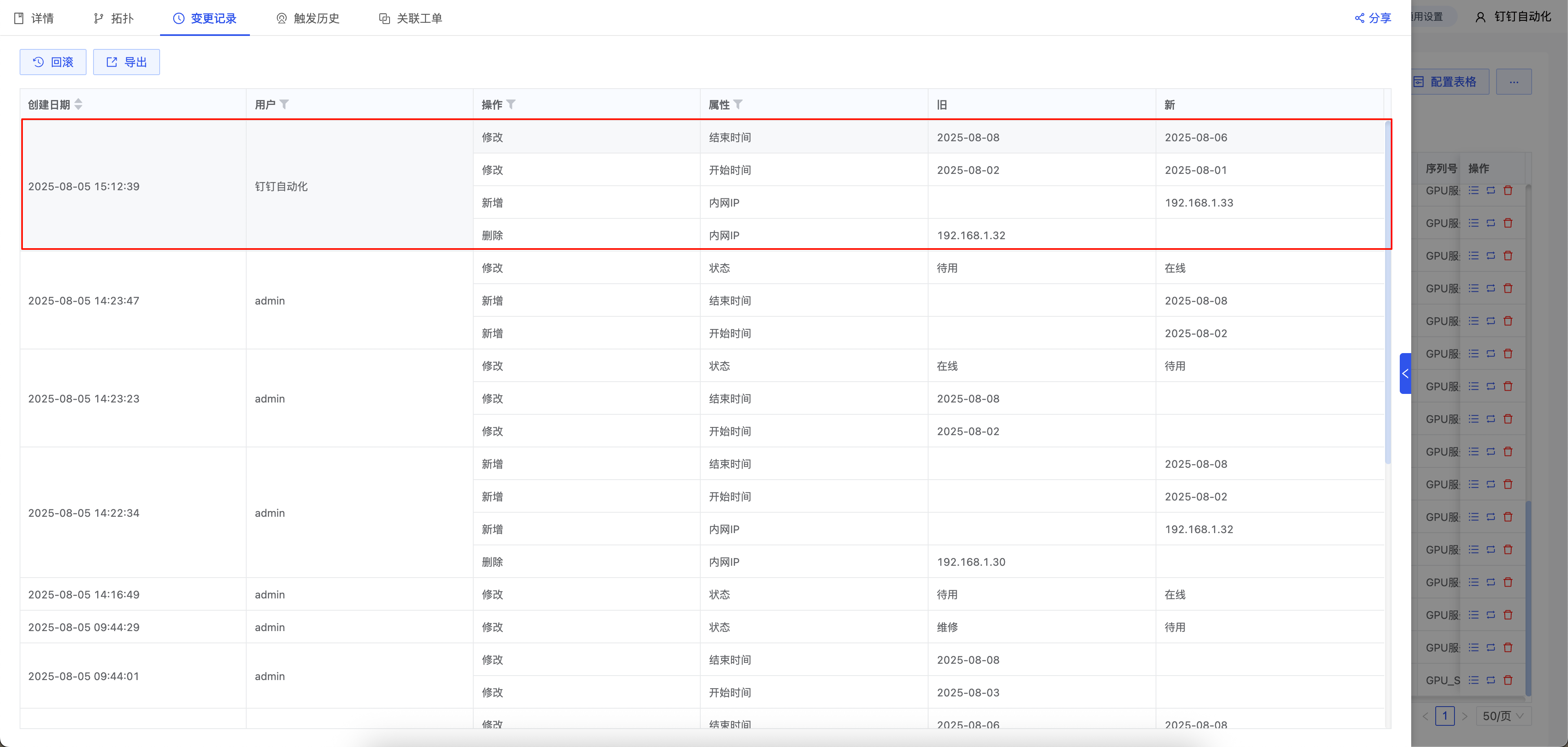
* 1. 接口细节
     1. 更新接口

|  |
| --- |
| curl -X 'POST' \  'http://10.1.100.45:8080/update-relation' \  -H 'accept: application/json' \  -H 'x-token: dingtalk-secret-token-327' \  -H 'Content-Type: application/json' \  -d '{  "sn": "GPU\_SERVER\_30",  "new\_project\_name": "钉钉调用的项目3",  "private\_ip": "192.168.1.33",  "s\_time": "2025-08-01",  "e\_time": "2025-08-06"  }' |

触发效果



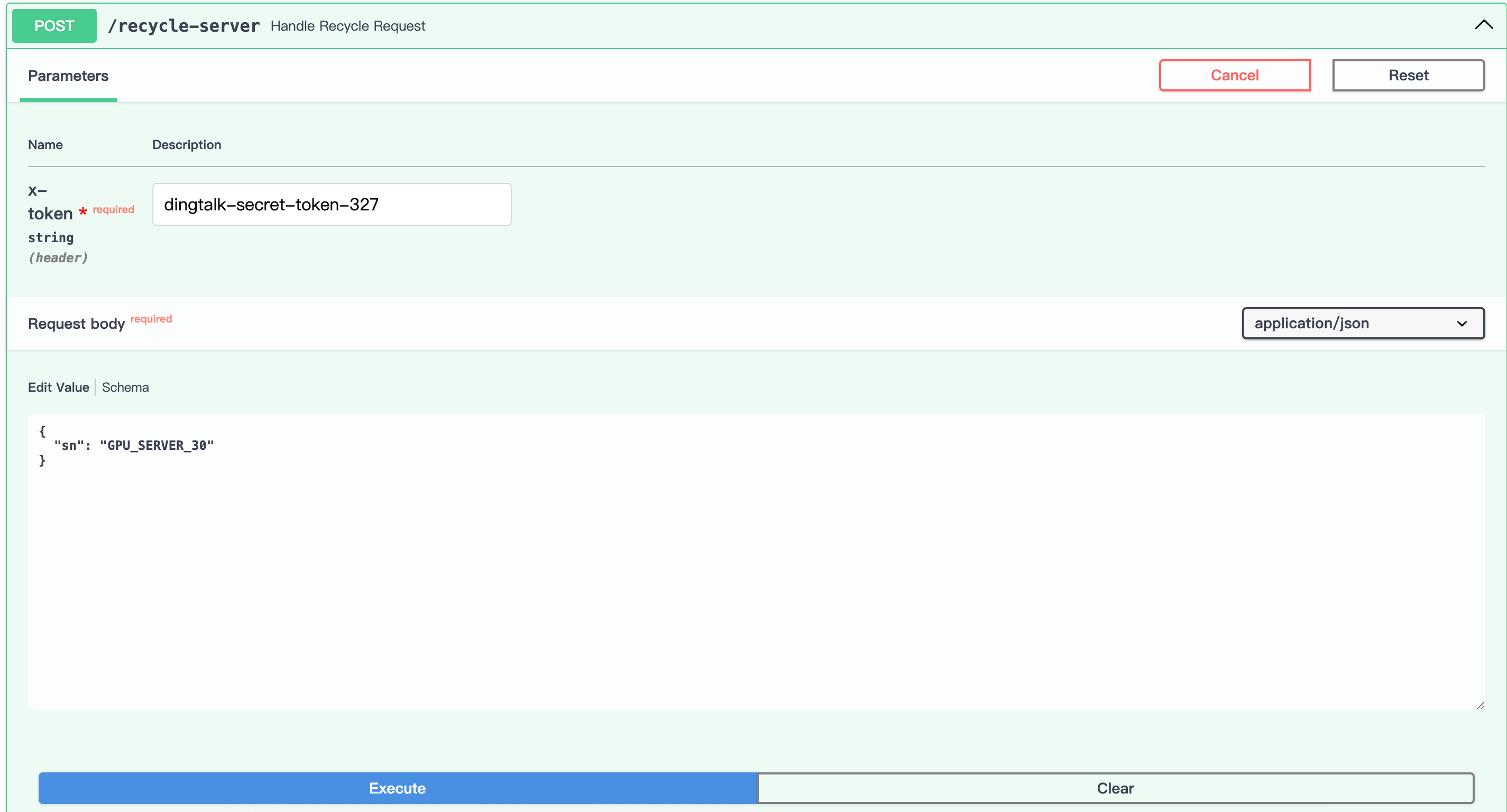


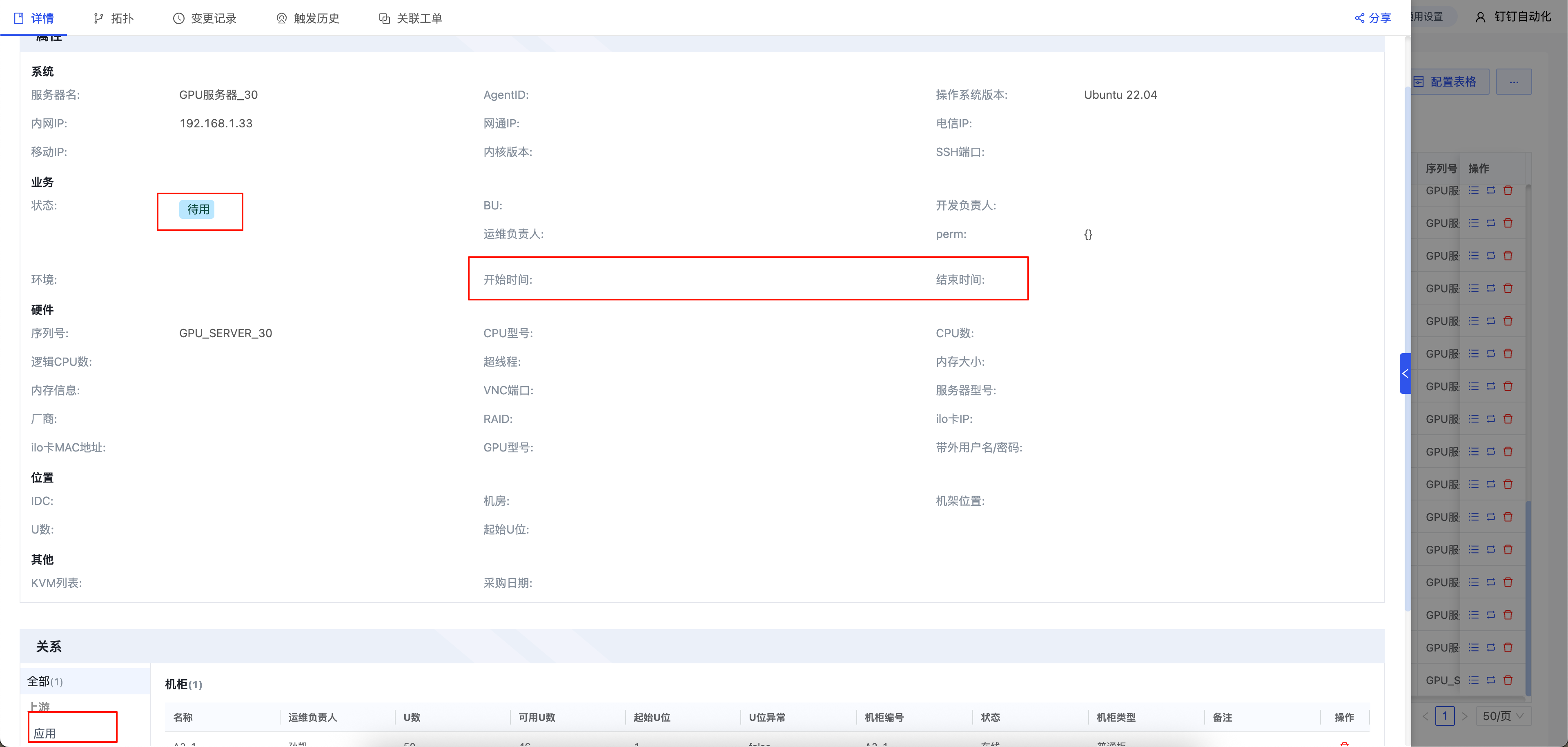


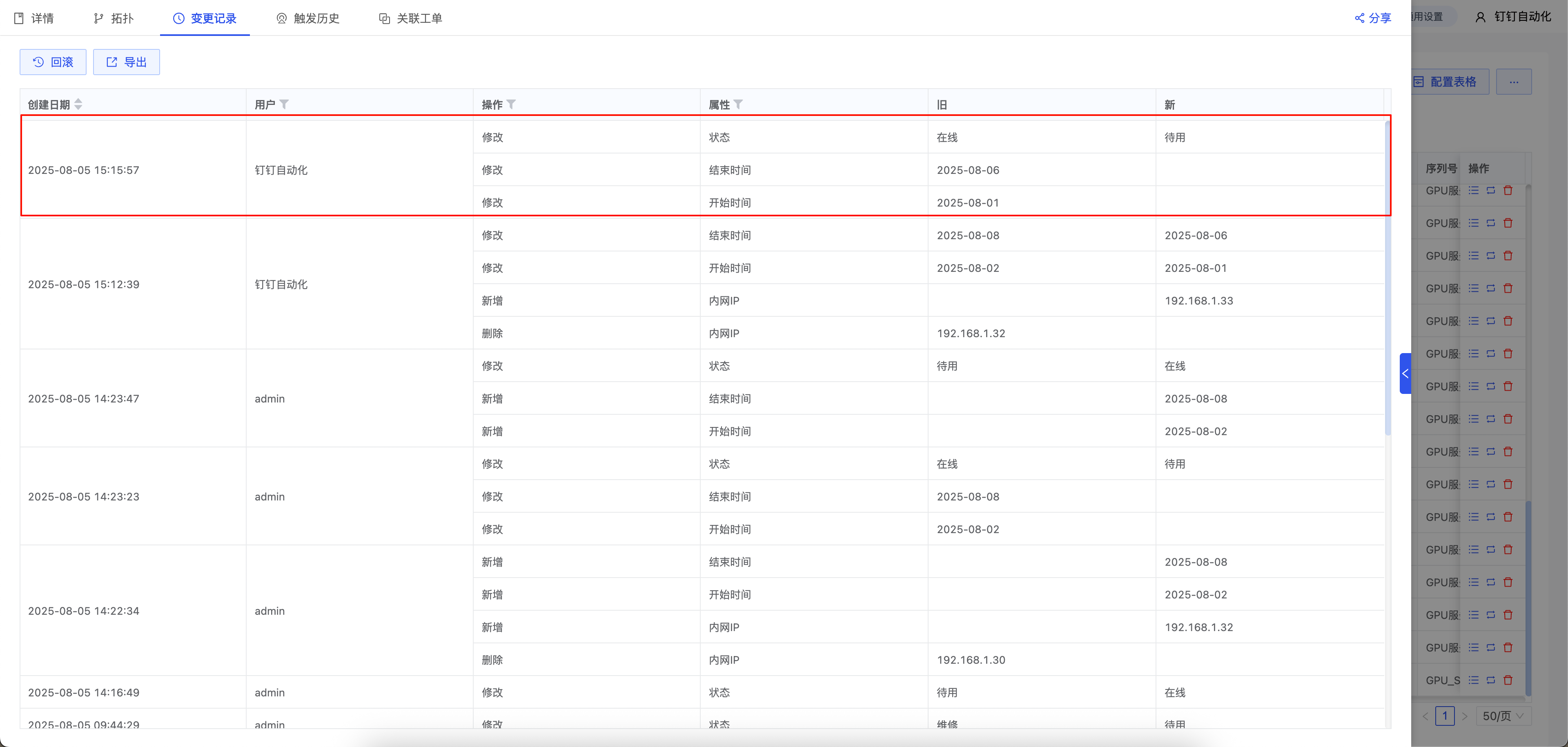
* + 1. 回收接口

|  |
| --- |
| curl -X 'POST' \  'http://10.1.100.45:8080/recycle-server' \  -H 'accept: application/json' \  -H 'x-token: dingtalk-secret-token-327' \  -H 'Content-Type: application/json' \  -d '{  "sn": "GPU\_SERVER\_30"  }' |

效果







* 1. 部署方式

docker镜像：registry.bitahub.com:5000/basic/cmdb-api:v1.7.0

|  |
| --- |
| docker run \  -d \  -p 8080:8080 \  --name cmdb-api-container \  --env CMDB\_URL="http://10.1.100.45:8000/api/v0.1" \  --env CMDB\_KEY="97aa21f39d674bea96288745a1dd4cec" \  --env CMDB\_SECRET='Dntzs^duhp4!mb2?c@~AweS#ZW\*QlYNx' \  --env API\_SECRET\_TOKEN="dingtalk-secret-token-327" \  registry.bitahub.com:5000/basic/cmdb-api:v1.7.0 |

* 1. 代码见附录

附录：

* 1. 单台物理机API创建

|  |
| --- |
| #!/usr/bin/env python3  import requests, hashlib  HOST = "http://10.1.100.45:8000"  KEY = "6783474dad5842589811b66492b3d2ae"  SEC = "OUEIneC5pgyG3ts01olxTbF#JVBX^vHf"  def sign(path, params):  kv = ''.join([str(params[k]) for k in sorted(params)])  return hashlib.sha1((path + SEC + kv).encode()).hexdigest()  payload = {  "ci\_type": "server",  "sn": "GPU\_SERVER\_30",  "server\_name": "GPU服务器\_30",  "status": "在线"  }  sig = sign("/api/v0.1/ci", payload)  payload.update(\_key=KEY, \_secret=sig)  r = requests.post(HOST + "/api/v0.1/ci", json=payload, timeout=5)  print(r.status\_code, r.text) |

* 1. CMDB-API源码

|  |
| --- |
| import os  from fastapi import FastAPI, HTTPException, Header  from pydantic import BaseModel  from typing import List, Optional  try:  from cmdb import Option, get\_client  except ImportError:  raise ImportError("无法导入 'cmdb' 模块。请确保 'cmdb-sdk-python' 已正确安装。")  # --- 1. FastAPI 应用配置 ---  # ==============================================================================  app = FastAPI(  title="CMDB 服务器管理 API",  description="用于更新服务器属性、管理项目关联以及回收服务器资源。",  version="1.7.0", # 增加状态更新功能  )  # --- 2. 从环境变量加载配置 ---  # ==============================================================================  CMDB\_URL = os.getenv("CMDB\_URL", "http://10.1.100.45:8000/api/v0.1")  CMDB\_KEY = os.getenv("CMDB\_KEY")  CMDB\_SECRET = os.getenv("CMDB\_SECRET")  API\_SECRET\_TOKEN = os.getenv("API\_SECRET\_TOKEN")  # --- 3. 定义 API 数据模型 (Pydantic) ---  # ==============================================================================  class ServerProjectRequest(BaseModel):  sn: str  new\_project\_name: str  private\_ip: Optional[str] = None  s\_time: Optional[str] = None  e\_time: Optional[str] = None  class RecycleRequest(BaseModel):  sn: str  class ApiResponse(BaseModel):  status: str  message: str  logs: List[str]  # --- 4. 核心业务逻辑函数 ---  # ==============================================================================  def update\_server\_project\_relation\_logic(  sn: str, new\_project\_name: str, private\_ip: Optional[str], s\_time: Optional[str], e\_time: Optional[str]  ) -> List[str]:  logs = []  if not all([CMDB\_URL, CMDB\_KEY, CMDB\_SECRET]):  logs.append("错误：服务器端未配置完整的CMDB连接信息！")  raise ValueError("CMDB credentials are not fully configured.")  opt = Option(url=CMDB\_URL, key=CMDB\_KEY, secret=CMDB\_SECRET)  cli = get\_client(opt)  server\_result = cli.get\_ci(q=f"\_type:server,sn:{sn}")  if not server\_result or not server\_result.result:  logs.append(f"错误：未找到服务器，SN: {sn}")  raise FileNotFoundError(f"Server with SN '{sn}' not found.")    server\_id = server\_result.result[0]['\_id']  logs.append(f"成功找到服务器 '{sn}'，CI ID: {server\_id}")  # --- 更新服务器属性 ---  # 1. 创建包含所有待更新字段的字典  update\_payload = {}  if private\_ip is not None:  update\_payload['private\_ip'] = private\_ip  if s\_time is not None:  update\_payload['s\_time'] = s\_time  if e\_time is not None:  update\_payload['e\_time'] = e\_time    # \*\*核心改动\*\*: 固定将服务器状态设置为 "在线"  update\_payload['status'] = '在线'  # 2. 调用 update\_ci (因为 status 总是存在，所以 update\_payload 不会为空)  logs.append(f"准备使用 update\_ci 更新属性: {update\_payload}")  try:  update\_resp = cli.update\_ci("server", ci\_id=server\_id, attrs=update\_payload)  logs.append(f"服务器CI属性更新请求已发送，响应: {getattr(update\_resp, 'message', 'OK')}")  except Exception as e:  error\_msg = f"错误：使用 update\_ci 更新服务器属性时失败: {e}"  logs.append(error\_msg)  raise ConnectionError(error\_msg)  # --- 关联关系处理部分 (无变化) ---  rels\_resp = cli.get\_ci\_relation(server\_id, reverse=1)  if rels\_resp and rels\_resp.result:  for rel in rels\_resp.result:  if rel.get("ci\_type") == "project":  old\_project\_name = rel.get("project\_name", "未知项目")  logs.append(f"找到当前关联项目: [{old\_project\_name}]，准备删除...")  delete\_resp = cli.delete\_ci\_relation(src\_ci\_id=rel["\_id"], dst\_ci\_id=server\_id)  logs.append(f"删除旧关联结果: {getattr(delete\_resp, 'message', 'OK')}")  break    project\_result = cli.get\_ci(q=f"\_type:project,project\_name:{new\_project\_name}")  new\_project\_id = None  if project\_result and project\_result.result:  new\_project\_id = project\_result.result[0]['\_id']  logs.append(f"成功找到目标项目 '{new\_project\_name}'，CI ID: {new\_project\_id}")  else:  logs.append(f"未找到项目 '{new\_project\_name}'，将自动创建...")  new\_project\_data = {  "project\_name": new\_project\_name, "project\_type": "裸金属租赁",  "project\_status": "租赁履约中", "mk\_duty": "胡飞",  "project\_description": f"为服务器 {sn} 自动创建的项目",  }  create\_resp = cli.add\_ci("project", new\_project\_data)  if hasattr(create\_resp, "ci\_id"):  new\_project\_id = create\_resp.ci\_id  logs.append(f"新项目 '{new\_project\_name}' 创建成功，CI ID: {new\_project\_id}")  else:  error\_msg = f"错误：创建新项目失败: {create\_resp}"  logs.append(error\_msg)  raise ConnectionError(error\_msg)    if new\_project\_id:  logs.append(f"准备将服务器 '{sn}' 关联到项目 '{new\_project\_name}'...")  add\_rel\_resp = cli.add\_ci\_relation(src\_ci\_id=new\_project\_id, dst\_ci\_id=server\_id)  if add\_rel\_resp:  logs.append("成功建立新的项目关联！")  else:  error\_msg = f"错误：建立新关联失败，响应为: {add\_rel\_resp}"  logs.append(error\_msg)  raise ConnectionError(error\_msg)    return logs  def recycle\_server\_logic(sn: str) -> List[str]:  logs = []  if not all([CMDB\_URL, CMDB\_KEY, CMDB\_SECRET]):  logs.append("错误：服务器端未配置完整的CMDB连接信息！")  raise ValueError("CMDB credentials are not fully configured.")  opt = Option(url=CMDB\_URL, key=CMDB\_KEY, secret=CMDB\_SECRET)  cli = get\_client(opt)  server\_result = cli.get\_ci(q=f"\_type:server,sn:{sn}")  if not server\_result or not server\_result.result:  logs.append(f"错误：未找到服务器，SN: {sn}")  raise FileNotFoundError(f"Server with SN '{sn}' not found.")    server\_id = server\_result.result[0]['\_id']  logs.append(f"成功找到待回收服务器 '{sn}'，CI ID: {server\_id}")    # --- 解除项目关联 (无变化) ---  rels\_resp = cli.get\_ci\_relation(server\_id, reverse=1)  project\_found = False  if rels\_resp and rels\_resp.result:  for rel in rels\_resp.result:  if rel.get("ci\_type") == "project":  project\_found = True  old\_project\_name = rel.get("project\_name", "未知项目")  logs.append(f"找到关联项目: [{old\_project\_name}]，准备解除关联...")  delete\_resp = cli.delete\_ci\_relation(src\_ci\_id=rel["\_id"], dst\_ci\_id=server\_id)  logs.append(f"解除关联结果: {getattr(delete\_resp, 'message', 'OK')}")  break  if not project\_found:  logs.append("服务器未关联任何项目，无需解除。")  # --- 更新服务器属性 ---  logs.append(f"准备清空服务器的时间字段并更新状态...")    # 1. 定义要清空和更新的字段  fields\_to\_update = {  "s\_time": "",  "e\_time": "",  "status": "待用" # \*\*核心改动\*\*: 固定将服务器状态设置为 "待用"  }    try:  # 2. 使用正确的 update\_ci 方法执行更新  update\_resp = cli.update\_ci("server", ci\_id=server\_id, attrs=fields\_to\_update)  logs.append(f"服务器字段更新请求已发送，响应: {getattr(update\_resp, 'message', 'OK')}")  except Exception as e:  error\_msg = f"错误：更新服务器字段时失败: {e}"  logs.append(error\_msg)  raise ConnectionError(error\_msg)  return logs  # --- 5. API 端点 (路由) ---  # (无变化)  def verify\_token(x\_token: str):  if not API\_SECRET\_TOKEN:  raise HTTPException(status\_code=500, detail="服务器端未配置API令牌 (API\_SECRET\_TOKEN is not set)")  if x\_token != API\_SECRET\_TOKEN:  raise HTTPException(status\_code=401, detail="无效的认证令牌 (Invalid Authentication Token)")  @app.post("/update-relation", response\_model=ApiResponse, tags=["Server Management"])  async def handle\_update\_request(payload: ServerProjectRequest, x\_token: str = Header(...)):  verify\_token(x\_token)  try:  logs = update\_server\_project\_relation\_logic(  payload.sn, payload.new\_project\_name,  payload.private\_ip, payload.s\_time, payload.e\_time  )  return {"status": "success", "message": f"服务器 '{payload.sn}' 的处理请求已成功完成。", "logs": logs}  except FileNotFoundError as e:  raise HTTPException(status\_code=404, detail=str(e))  except (ValueError, ConnectionError, Exception) as e:  raise HTTPException(status\_code=500, detail=f"处理请求时发生内部错误: {e}")  @app.post("/recycle-server", response\_model=ApiResponse, tags=["Server Management"])  async def handle\_recycle\_request(payload: RecycleRequest, x\_token: str = Header(...)):  verify\_token(x\_token)  try:  logs = recycle\_server\_logic(payload.sn)  return {"status": "success", "message": f"服务器 '{payload.sn}' 已成功回收。", "logs": logs}  except FileNotFoundError as e:  raise HTTPException(status\_code=404, detail=str(e))  except (ValueError, ConnectionError, Exception) as e:  raise HTTPException(status\_code=500, detail=f"处理请求时发生内部错误: {e}")  @app.get("/", tags=["Health Check"], include\_in\_schema=False)  async def root():  return {"message": "CMDB Server Management API is running."} |

直接运行脚本测试方式

设置环境变量

在终端中，进入 main.py 所在目录，并执行以下命令来设置环境变量。

Generated bash

export CMDB\_URL="http://10.1.100.45:8000/api/v0.1"

export CMDB\_KEY="6783474dad5842589811b66492b3d2ae"

export CMDB\_SECRET="OUEIneC5pgyG3ts01olxTbF#JVBX^vHf"

export API\_SECRET\_TOKEN="dingtalk-secret-token-12345" Use code with caution.

启动服务

在同一个终端中，使用 uvicorn 启动服务。

# --host 0.0.0.0 让服务可以被局域网内的其他机器（比如钉钉服务器）访问

# --port 8080 指定一个端口

uvicorn main:app --reload --host 0.0.0.0 --port 8080

--reload 选项表示当你修改代码并保存后，服务会自动重启

测试服务

服务启动后，你有两种方式测试：

方法A: 使用自动生成的交互式文档 (推荐)

打开浏览器，访问 http://10.1.100.41:8080/docs。

你会看到一个漂亮的 API 文档界面。

点击 /update-relation 端点，然后点 "Try it out"。

在 X-Token 字段中输入你设置的令牌 dingtalk-secret-token-12345。

在 Request body 中输入 JSON 数据，例如:

Generated json

{

"sn": "GPU\_SERVER\_30",

"new\_project\_name": "钉钉调用的项目"

}

Use code with caution.

Json

点击 "Execute"，你就可以看到服务返回的实时响应了。