Hw-Link 物联网云平台

使用手册

Ver3.0

宏微自动化软件工作室

http://www.hwzdh.cn

目录

一、关于 Hw-Link 物联网云平台

简介

Hw-Link Web 组态软件采用 Vue2+Go 语言开发，通过浏览器操作组态工具、浏览组态画面，实现工程管理、组态编辑、工业设备采集以及组态运行三大功能。通过实现图元组态、可视化图表组态、数据库组态的配置与关联，完成基于 Web 服务的实时数据监控与服务端的多用户访问等。采用标准 HTML5 技术，基于 B/S 架构进行开发，支持 WEB 端呈现，支持在浏览器端完成便捷的人机交互，简单的拖拽即可完成可视化页面的编排设计。支持 Windows、Centos 等其他基于 Linux 内核的系统。具有安装简单、操作方便、功能实用等特点。

官方网站: http://www.hwzdh.cn/

体验账户购买：https://item.taobao.com/item.htm?ft=t&id=949940898254

愿景

打造国内免费组态软件新篇章。愿所有中小企业都能使用到免费组态软件，赋能数据可视化。

提供价值

为企业可视化赋能
高效快捷的可视化工具，把无形的数据变成有意义、清晰可见的展示；
数据驱动显示，让我们更专注业务本身，提高生产力；
0 代码就可以完成实时的数据采集、数据动效。

二、Hw-Link 软件下载与安装

下载

打开 http://www.hwzdh.cn/download.html 网站，点击下载按钮，跳转到下载界面，根据自己的需要下载相应的版本
支持 windows、Centos 系统

Windows 安装

打开下载地址:安装包下载地址下载安装包 Hw-Link-*.zip *表示版本号,下载完成后，双击压缩包里面的 setup.exe，按照提示步骤，一步步安装。

登录

在浏览器中输入 http://127.0.0.1:8081/ 如果是局域网访问,请输入 http://局域网 IP:8081,比如：http://192.168.1.162:8081 即可打开 Hw-Link 组态软件，支持谷歌浏览器、edge 等基于 Chromium 内核浏览器，注意: 如果系统打开了防火墙，局域网访问的时候，要把防火墙关掉或者允许 8081 通过，默认的登录密码:
登录用户名：admin
登录密码：99999999（8 个 9）

三、Hw-Link 软件的配置文件说明

配置文件路径

在安装目录 conf/app.conf，默认不需要配置，用户可根据现场的场景修改。

appname=ISM #应用名称
mysqldbname=ism #MySQL 数据库数据库名称
sessionprovider=file #session 的存储方式，不可更改
enableerrorsshow=true #不可更改
mysqluser=root #MySQL 的用户名
customdatadealwithdelaytime=100 #自定义数据的推送时间，单位毫秒
servername=ISM #应用服务名称
httpport=8081 #Hw-Link 的 Web 端口
sessiongcmaxlifetime=36000 #session 的保存时间
staticextensionstogzip=.css, .js, .png, .jpg, .ico, .mp4, .lic, .ttf #开启 Gzip 压缩后，配置压缩的文件后缀，配置后就可以只用 Gzip 压缩，节省加载时间
directoryindex=false
runmode=prod
mysqlpwd=Crazy1234567 #MySQL 的密码
graceful=true
dbtype=1 #数据库类型 1 是 sqlite 0 是 MySQL
sessionon=true
jwtkey=4ti7ng2y0u
copyrequestbody=true
staticdatadealwithdelaytime=100
sessionproviderconfig=./data/sessionon
sessionsookielifetime=36000
tokenexpirestime=744 #token 过期时间 ，单位小时
recoverpanic=true
sendalarmspeed=1000
mysqlhost=16.100.1.154 #mysql 服务器 IP，不用手动修改，可以在界面上配置
sessionname=ISMSession
mysqlport=3308 #mysql 服务器的端口，不用手动修改，可以在界面上配置
logfilessavadays=5 #日志保存天数，超出此天数后自动删除旧的日志文件
EnableGzip=true #是否启用 Gzip 压缩，默认开启，可以节省加载时间
gzipCompressLevel = 2 #Gzip 压缩等级，默认 2，不建议修改
EnableErrorsShow = true #是否显示错误信息，不建议修改
gzipMinLength = 256 #不建议修改
includedMethods = get;post #不建议修改
WSPort=10215 #websocket 端口,用于与前端和组态界面通信
WSAddress="local" #websocket 代理地址，如果局域网里面启用代理，此处赢填

写真实的 IP 地址

enablehttps=false #是否开启 HTTPS 协议支持，true 是支持，false 不支持，默认 false
httpsport=10443 #HTTPS 端口 ，默认 10443
httpskeyfile=conf/192.168.199.120.key #HTTPS 的证书加密密码
httpscertfile=conf/192.168.199.120.crt #HTTPS 的证书

四、Hw-Link 端口配置和说明

4.1 功能说明

下面提到的端口需要在系统的防火墙或者路由器中添加相关规则，把这些端口开放出去。一定要注意 10215 端口，否则数据仓库中看不到数据更新。

4.2 简介

Hw-Link 系统使用了多个端口用于数据通信，下面一一说明。

4.3 Web 端口 8081

Hw-Link 使用了默认的 8081 端口提供了 Web 服务，此端口可以在配置文件中修改

4.4 websocket 端口默认 10215

Hw-Link 页面使用 websocket 与 Hw-Link 后台数据通信，用于前端页面的数据展示和组态界面的数据查看。

4.5 Modbus Server 端口 3000

Hw-Link 通过 3000 端口与 DTU 通信，用于采集 modbus 设备数据

4.6 DLT645 Server 端口 10216

Hw-Link 通过 10216 端口与 DTU 通信，用于采集 DLT645 电表数据

4.7 WebRtc 端口默认 50010 到 50100

视频在线播放的 webrtc 端口从 50010 到 50100，此端口可以在 conf 目录下面的 videoConfig.json 中的 webrtc_port_min 和 webrtc_port_max 中修改

五、数据库配置

5.1 涛思数据库安装与配置

5.1.1 功能说明

为了更大规模的存储数据，Hw-Link 支持把采集的数据按规则存储到数据库中。支持涛思时序数据库
配置文件在安装目录下面的 conf 文件下面的 historyData.conf 配置文件，按照下面的配置的说明，配置相关的参数
注意:Hw-Link 只会把历史数据存储到时序数据库中，Hw-Link 运行需要的参数还是保存在 sqlite 或者 mysql 数据库中

5.1.2 安装涛思数据库

TDengine 完整的软件包包括服务端（taosd）、应用驱动（taosc）、用于与第三方系统对接并提供 RESTful 接口的 taosAdapter、命令行程序（CLI，taos）和一些工具软件。目前 TDinsight 仅在 Linux 系统上安装和运行，后续将支持 Windows、macOS 等系统。TDengine 除了提供多种语言的连接器之外，还通过 taosAdapter 提供 RESTful 接口。
下载地址
版本选择 3.0.7.0
https://www.taosdata.com/assets-download/3.0/TDengine-server-3.0.7.0-Windows-x64.exe

安装完成后，双击 Taos Shell，运行 taosd.exe

再次双击 Taos Shell，运行 taosadapter.exe 即可启动涛思数据库

5.1.3 配置涛思数据库

在安装目录下面的 conf 文件里面找到 historyData.conf 配置文件

把字段的值修改为 2 ，HistoryRecordDbType=2

TDengine 字段后面的按照安装的涛思数据库配置填写相关参数

```
HistoryRecordDbType=2
[TDengine]
TDenginePort=6041                    #涛思数据库的端口
TDengineHost="127.0.0.1"             #涛思数据库地址
UserName="root"                      #涛思数据库的用户名
PassWord="taosdata"                  #涛思数据库的用户的密码
```

然后重启 Hw-Link 即可，注意观察 Hw-Link 是否能连接上，如果连接不成功，请根据提示配置涛思数据库

5.2 Influxdb 数据库安装与配置

5.2.1 功能说明

为了更大规模的存储数据，Hw-Link 支持把采集的数据按规则存储到数据库中。支持涛思时序数据库

配置文件在安装目录下面的 conf 文件下面的 historyData.conf 配置文件，按照下面的配置的说明，配置相关的参数

注意:Hw-Link 只会把历史数据存储到时序数据库中，Hw-Link 运行需要的参数还是保存在 sqlite 或者 mysql 数据库中

5.2.2 参考

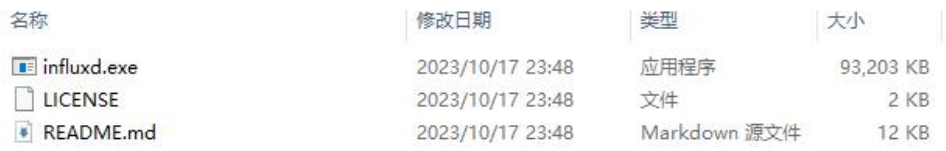
官方文档：https://docs.influxdata.com/influxdb/v2/

5.2.3 下载

需要下载两样东西：influxd.exe 和 influx.exe

influxd:influx 数据库的服务端。下载地址：https://dl.influxdata.com/influxdb/releases/influxdb2-2.7.5-windows.zip

下载后解压，都放到一个文件夹里：

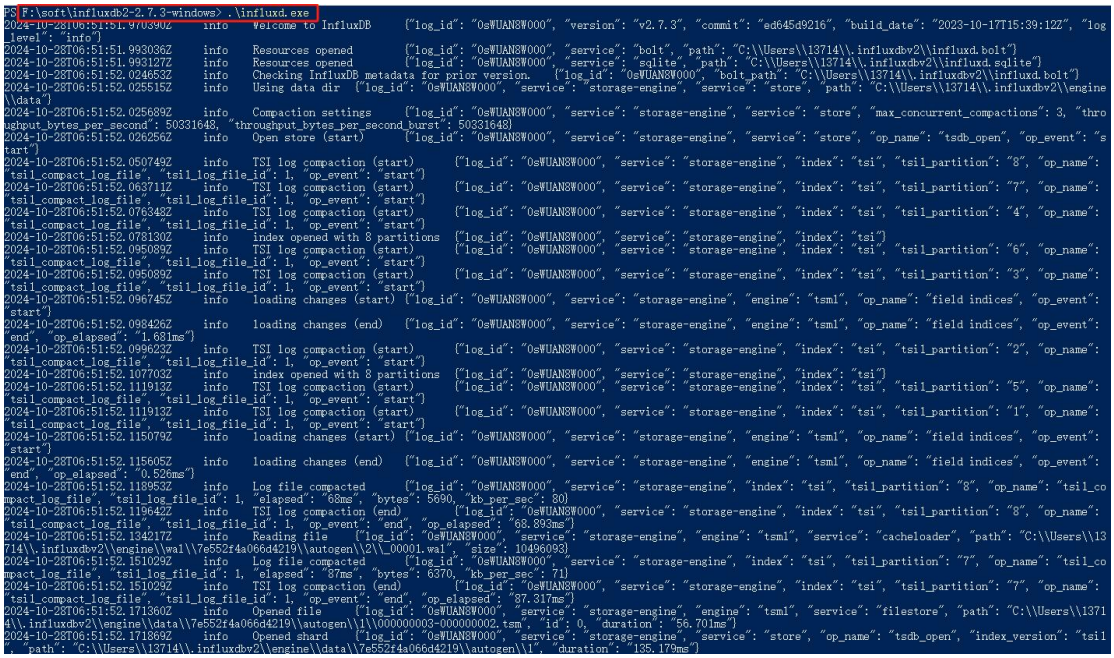


名称	修改日期	类型	大小
influxd.exe	2023/10/17 23:48	应用程序	93,203 KB
LICENSE	2023/10/17 23:48	文件	2 KB
README.md	2023/10/17 23:48	Markdown 源文件	12 KB

5.2.4 启动

默认配置启动

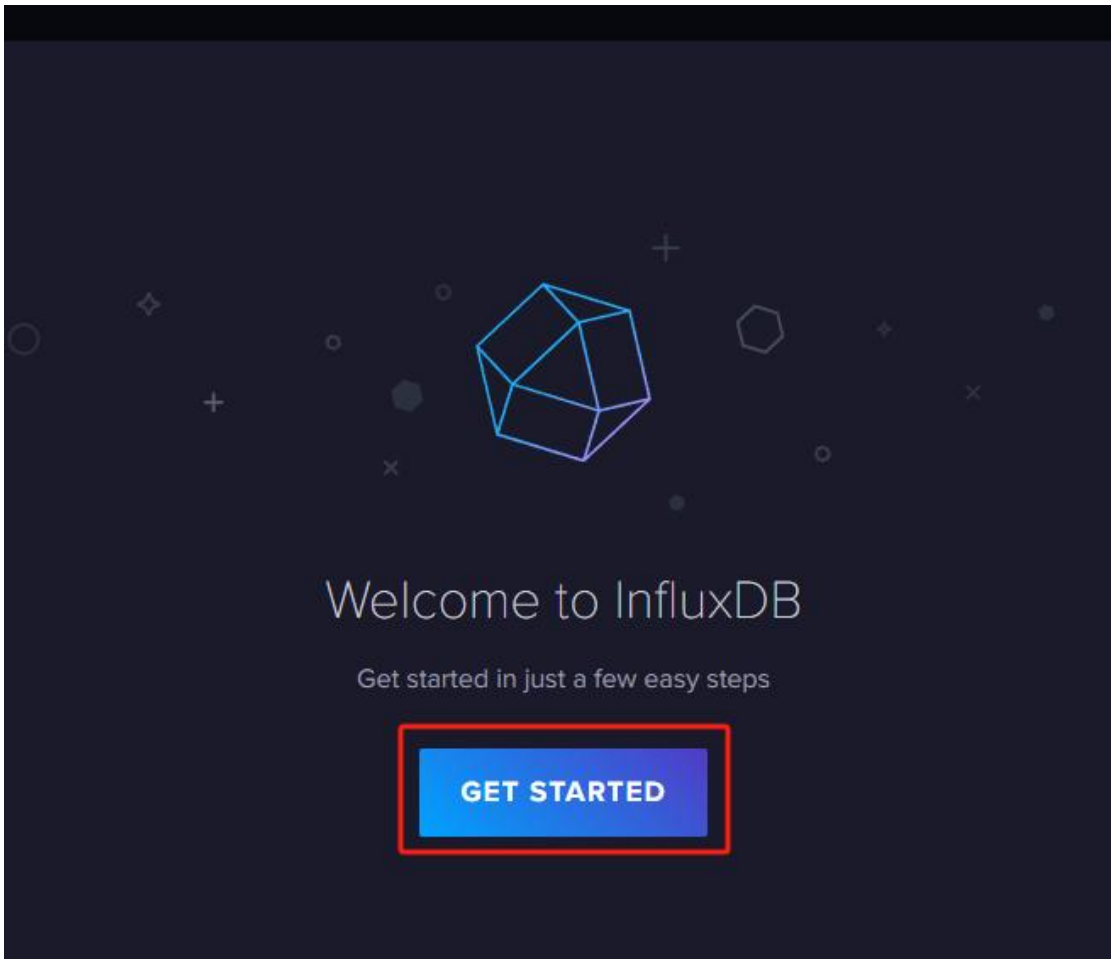
在该目录下打开命令行窗口，输入 influxd.exe 即可启动：



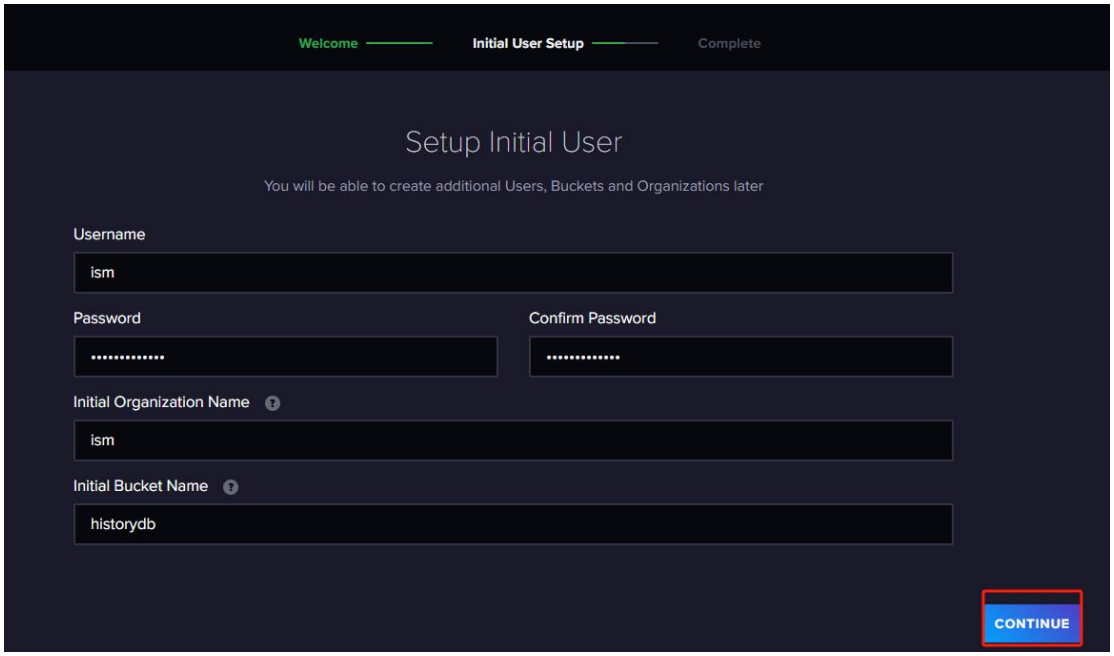
存储数据的地方是默认的(C:\Users\29438.influxdbv2)

5.2.5Token 和 Bucket

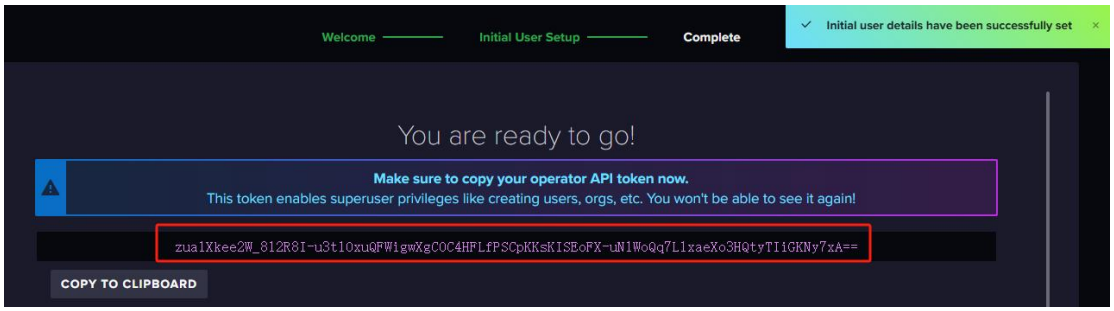
打开网址：http://localhost:8086



配置 Initial Organization Name 和 Initial Bucket Name，点击继续



完成后会提示你需要的 token,复制此 token



详细的文档请参考: https://blog.csdn.net/lishuangquan1987/article/details/137401045

5.2.6 配置 Influxdb 数据库

在安装目录下面的 conf 文件里面找到 historyData.conf 配置文件
把字段的值修改为 4 ， HistoryRecordDbType=4
Influxdb 字段后面的按照安装的 Influxdb 数据库配置填写相关参数
HistoryRecordDbType=4
[Influxdb]
Url=http://127.0.0.1:8086
Token=ytjVFYEKoHWBZk8fgdHPv-QbYuXNCFZLPTZs1sPKy_SAp78sCYqq96dISx5u73CPRI1tBZcrwhIzDP
OW5pKxXw==
Org="ism"
Bucket="ism"
然后重启 Hw-Link 即可，注意观察 Hw-Link 是否能连接上，如果连接不成功，请根据提示配置 Influxdb 数据库

5.3 ClickHouse 数据库安装与配置

5.3.1ClickHouse 的特性

在一个真正的列式数据库管理系统中，除了数据本身外不应该存在其他额外的数据。这意味着为了避免在值旁边存储它们的长度«number»，你必须支持固定长度数值类型。例如，10 亿个 UInt8 类型的数据在未压缩的情况下大约消耗 1GB 左右的空间，如果不是这样的话，这将对 CPU 的使用产生强烈影响。即使是在未压缩的情况下，紧凑的存储数据也是非常重要的，因为解压缩的速度主要取决于未压缩数据的大小。
这是非常值得注意的，因为在一些其他系统中也可以将不同的列分别进行存储，但由于对其他场景进行的优化，使其无法有效的处理分析查询。例如： HBase，BigTable，Cassandra，HyperTable。在这些系统中，你可以得到每秒数十万的吞吐能力，但是无法得到每秒几亿行的吞吐能力。
需要说明的是，ClickHouse 不单单是一个数据库， 它是一个数据库管理系统。因为它允许在运行时创建表和数据库、加载数据和运行查询，而无需重新配置或重启服务。

5.3.2 系统要求

ClickHouse 可以在任何具有 x86_64，AArch64 或 PowerPC64LE CPU 架构的 Linux，FreeBSD 或 Mac OS X 上运行。官方预构建的二进制文件通常针对 x86_64 进行编译，并利用 SSE 4.2 指令集，因此，除非另有说明，支持它的 CPU 使用将成为额外的系统需求。下面是检查当前 CPU 是否支持 SSE 4.2 的命令:

```
$ grep -q sse4_2 /proc/cpuinfo && echo "SSE 4.2 supported" || echo "SSE 4.2 not supported"
```

要在不支持 SSE 4.2 或 AArch64，PowerPC64LE 架构的处理器上运行 ClickHouse，您应该通过适当的配置调整从源代码构建 ClickHouse。

5.3.3DEB 安装包

建议使用 Debian 或 Ubuntu 的官方预编译 deb 软件包。运行以下命令来安装包:

```
sudo apt-get install -y apt-transport-https ca-certificates dirmngr
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 8919F6BD2B48D754
echo "deb https://packages.clickhouse.com/deb stable main" | sudo tee \
    /etc/apt/sources.list.d/clickhouse.list
sudo apt-get update
sudo apt-get install -y clickhouse-server clickhouse-client
sudo service clickhouse-server start
```

5.3.4RPM 安装包

推荐使用 CentOS、RedHat 和所有其他基于 rpm 的 Linux 发行版的官方预编译 rpm 包。

首先，您需要添加官方存储库：

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://packages.clickhouse.com/rpm/clickhouse.repo
sudo yum install -y clickhouse-server clickhouse-client

sudo /etc/init.d/clickhouse-server start
```

5.3.5 其他方式安装

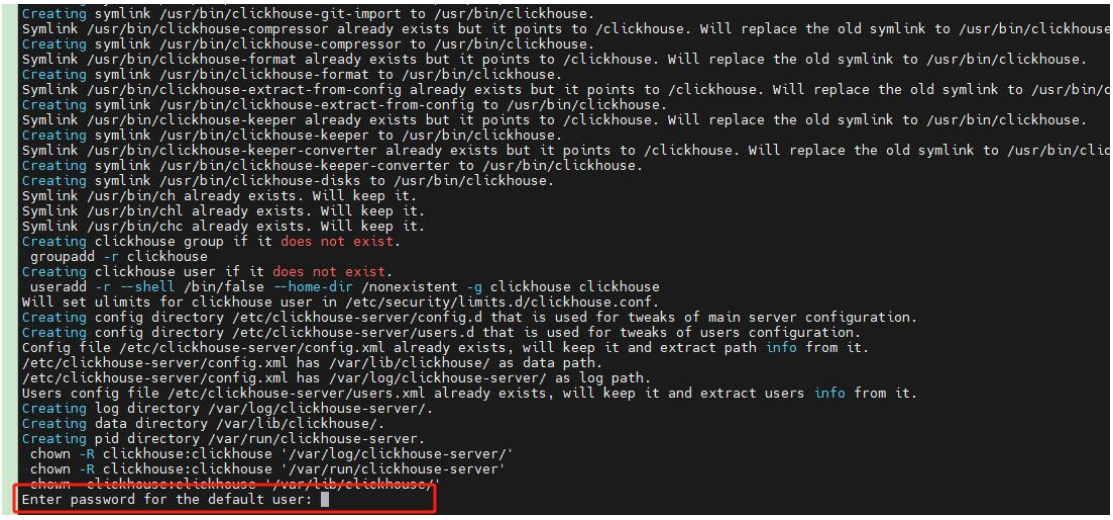
其他方式安装，请参考官方文档

https://clickhouse.com/docs/zh/getting-started/install

5.3.6 添加密码

clickhouse 默认提供了密码加密传输的方式，这样比明文传输要安全的多。

安装时会提示你输入用户 default 的密码，



输入密码回车后，即可完成安装安装

输入 service clickhouse-server restart 重启服务

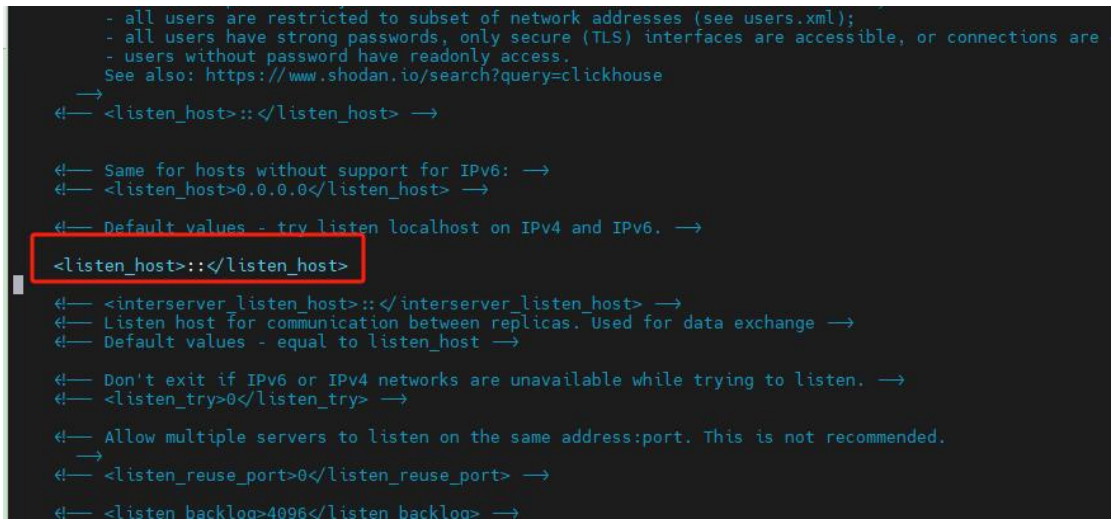
service clickhouse-server restart

5.3.7 远程访问

编辑 /etc/clickhouse-server/config.xml 配置文件

vim /etc/clickhouse-server/config.xml

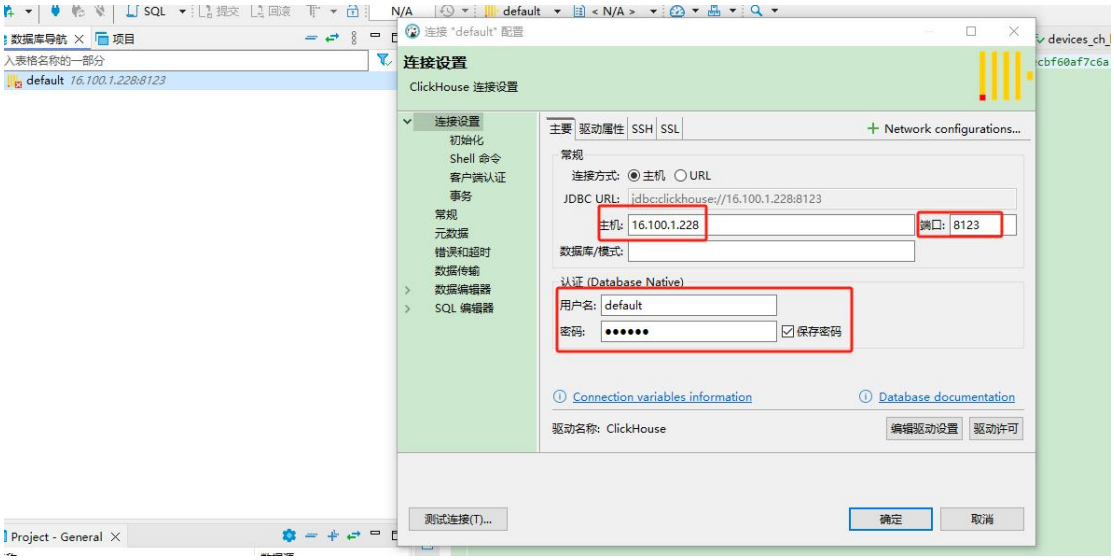
找到 listen_host 这个参数修改为下图所示



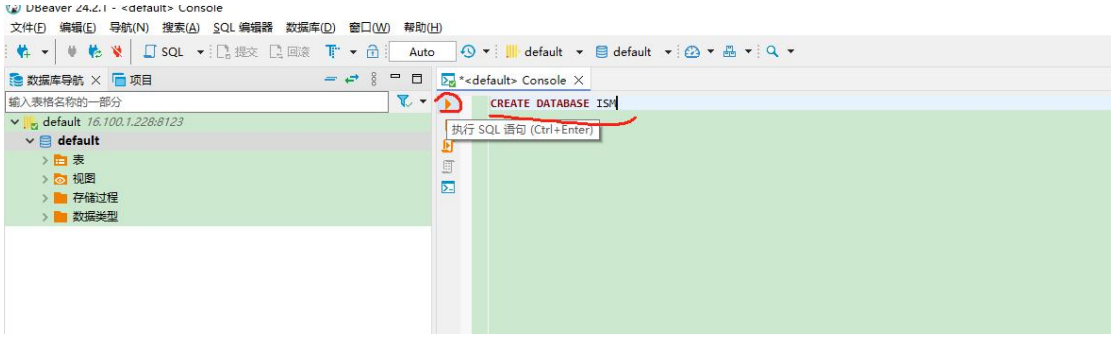
输入 service clickhouse-server restart 重启服务

service clickhouse-server restart

使用 DBeaver 管理工具连接，连接 clcikhouse



连接成功后，打开 SQL 控制台，再控制台里面输入 CREATE DATABASE ISM，创建 Hw-Link 数据库



5.3.8 配置 ClickHouse 数据库

在安装目录下面的 conf 文件里面找到 historyData.conf 配置文件

把字段的值修改为 3 ， HistoryRecordDbType=3

ClickHouse 字段后面的按照安装的 ClickHouse 数据库配置填写相关参数

HistoryRecordDbType=3

[ChickHouse]

ChickHousePort=9000

ChickHouseHost=16.100.1.228

UserName=default

PassWord=000000

DataBase=ISM

ConnectTimeout=60s

ReadTimeout=60s

然后重启 Hw-Link 即可，注意观察 Hw-Link 是否能连接上，如果连接不成功，请根据提示配置 ClickHouse 数据库

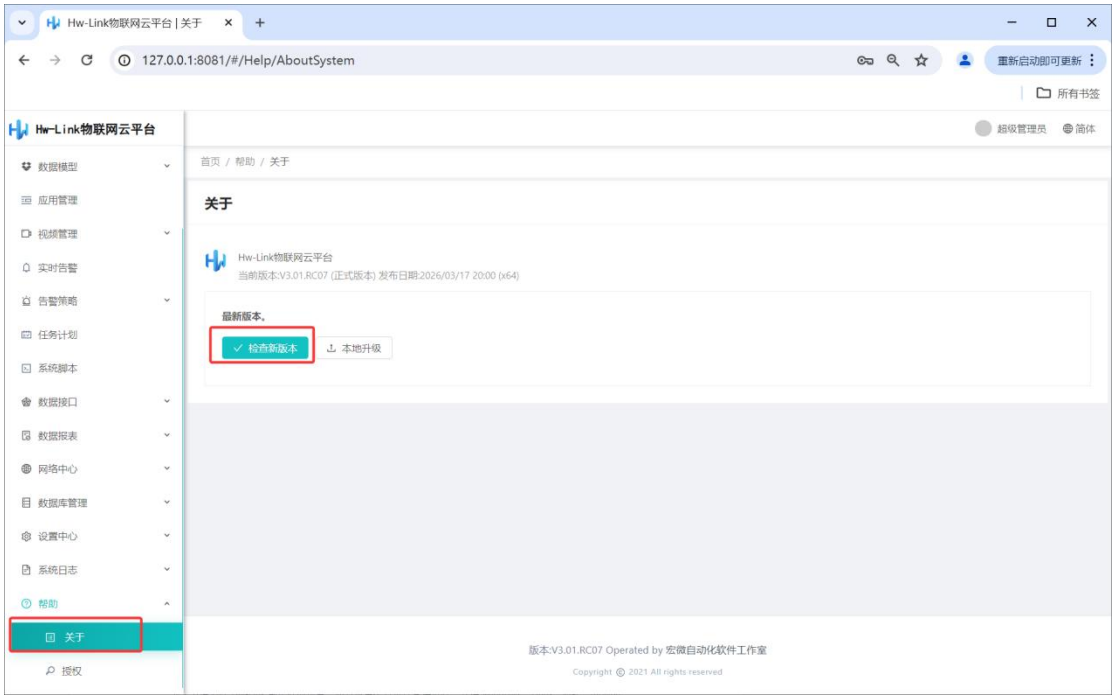
六、软件升级

功能说明

Hw-Link 提供了在线和离线两种升级方式。并且这两种方式都是免费的。

6.1 在线升级

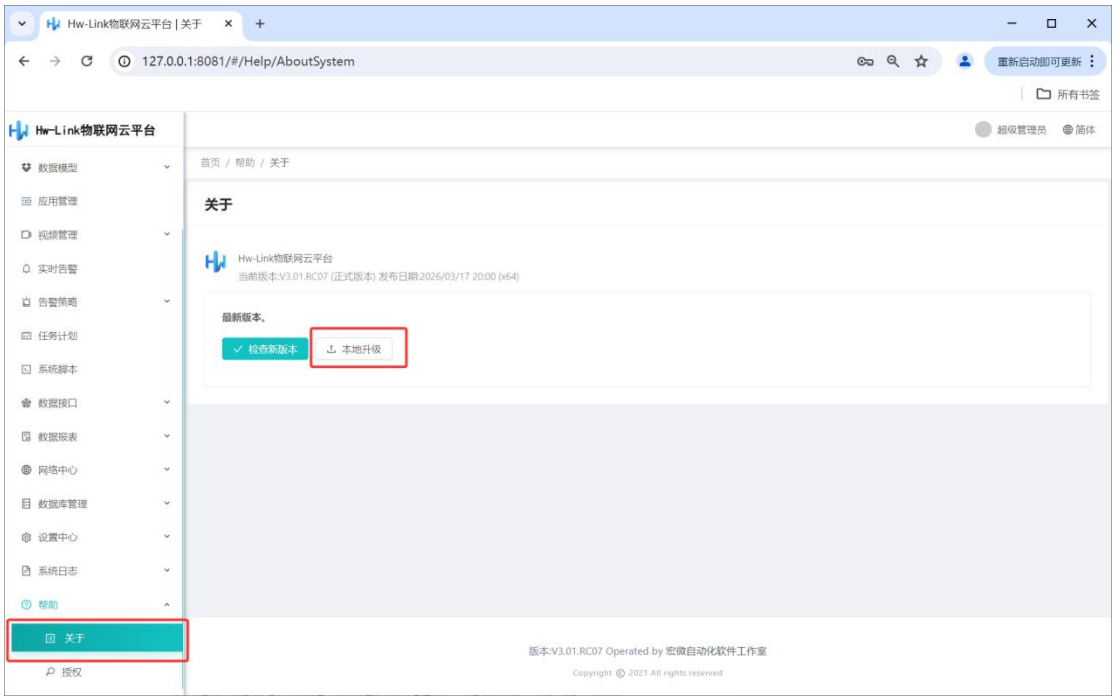
点击检查新版本，如果有的新的版本会提示用户是否升级，单击升级 Hw-Link 就会主动下载升级文件，然后升级



6.2 本地升级

本地升级是为了内网用户，没有办法在线升级的情况下可以使用。

点击本地升级，然后选择下载下来的升级包，Hw-Link 同样也会主动完成升级。


Hw-Link物联网云平台 | 关于
127.0.0.1:8081/#/Help/AboutSystem
重新启动即可更新
所有书签
Hw-Link物联网云平台
超级管理员
简体
数据模型
应用管理
视频管理
实时告警
告警策略
任务计划
系统脚本
数据接口
数据报表
网络中心
数据库管理
设置中心
系统日志
帮助
关于
授权
首页 / 帮助 / 关于
关于
Hw-Link物联网云平台
当前版本:V3.01.RC07 (正式版本) 发布日期:2026/03/17 20:00 (x64)
最新版本。
检查新版本
本地升级
版本:V3.01.RC07 Operated by 宏微自动化软件工作室
Copyright © 2021 All rights reserved

七、打包运行组态界面

功能说明

为了方便运行 Hw-Link 制作好的应用，通过应用配置可以直接运行，支持 Windows、Linux、Mac、ARM64。

7.1 下载运行程序

打开下载地址:安装包下载地址

下载自己需要的运行的版本，支持 windows、Linux、ARM。

7.2 配置及启动

解压后，目录结构如下:

名称	修改日期	类型	大小
locales	2024/10/29 9:31	文件夹	
resources	2024/10/29 9:31	文件夹	
chrome_100_percent.pak	2023/12/22 17:37	PAK 文件	134 KB
chrome_200_percent.pak	2023/12/22 17:37	PAK 文件	193 KB
chrome_crashpad_handler	2023/12/22 17:37	文件	1,254 KB
chrome-sandbox	2023/12/22 17:37	文件	53 KB
icudtl.dat	2023/12/22 17:37	dat file	10,383 KB
ismshowapp	2023/12/22 17:37	文件	164,677 KB
libEGL.so	2023/12/22 17:37	SO 文件	247 KB
libffmpeg.so	2023/12/22 17:37	SO 文件	2,815 KB
libGLESv2.so	2023/12/22 17:37	SO 文件	6,349 KB
libvk_swiftshader.so	2023/12/22 17:37	SO 文件	4,484 KB
libvulkan.so.1	2023/12/22 17:37	1 文件	6,992 KB
LICENSE.electron.txt	2023/12/22 17:37	Notepad++ Doc...	2 KB
LICENSES.chromium.html	2023/12/22 17:37	HTML 文件	8,649 KB
resources.pak	2023/12/22 17:37	PAK 文件	5,289 KB
snapshot_blob.bin	2023/12/22 17:37	BIN 文件	263 KB
v8_context_snapshot.bin	2023/12/22 17:37	BIN 文件	582 KB
vk_swiftshader_icd.json	2023/12/22 17:37	Notepad++ Doc...	1 KB

找到 ismshowapp 这个文件，这个程序启动文件，运行此程序即可，首先要赋予此文件可执行权限。

chmod 777 ismshowapp

在解压目录里面新建 AppRunConfig.json 文件，文件内容为

```
{
        "RunAppUrl":"http://16.100.1.230:8081/#/",
        "height":1024,
        "width":900,
        "icoPath":"./app.ico",
        "fullscreenable":true
}
```

7.3 配置文件说明

RunAppUrl：为运行的 Hw-Link 应用的地址，例如：

http://127.0.0.1:8081/#/AppRun/56a170d7-1731-cfeb-52f1-873a6d16e690

height：窗口的高度

width：窗口的宽度

icoPath:"应用图标路径" 暂时无用

fullscreenable：是否全屏，true 开启全屏，false 关闭全屏

运行后，启动画面等待 10s 即可

启动画面

八、数据采集

8.1 设备数据前言

首先我们要理解数据模型的概念，然后才能去创建我们的设备数据模型。

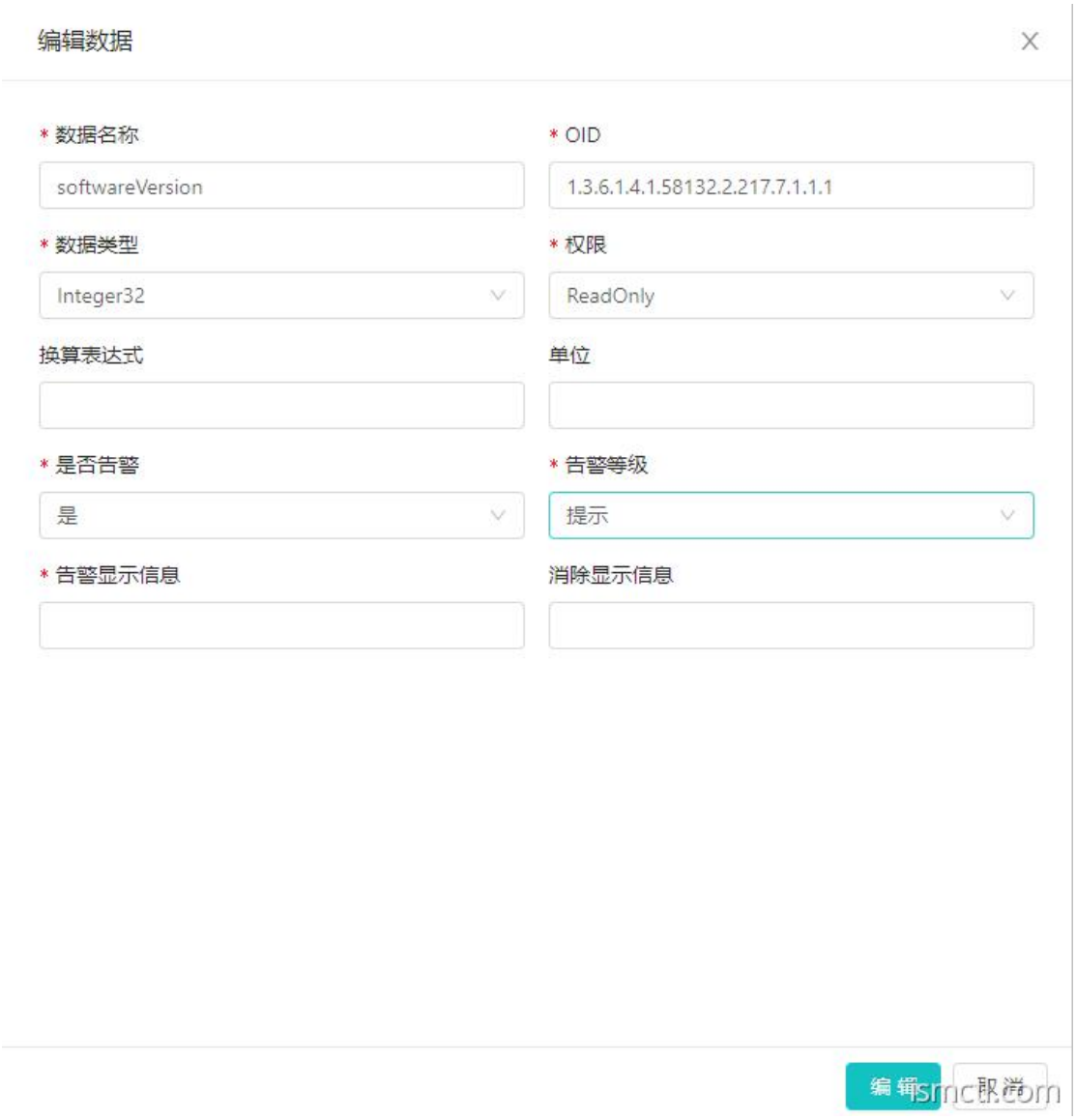
数据模型，说白了就是设备共有属性，比如说有个设备具有 10 条数据属性，那么跟这个相同设备也同样具有设备的属性，所以我们只需要创建一个数据

属性，就可以对应多个设备。任何数据属性的修改都可以同步到设备中去。这样就很简单的创建了一个数据模型，来对应多个设备。

Hw-Link 正是基于此理念，开发了设备数据模型->设备->设备组态应用。只要理解了这个下面的设备的创建，设备数据修改都会很顺手了。

8.2 设备数据的共有属性

每个设备模型都具有，数据类型、权限、换算表达式、单位、是否告警、是否存储等共有的属性，下面我将一一讲解。



换算表达式: 举个例子，Hw-Link 采集数据我们想通过简单的运算，才能真实的展示设备的数据，这时我们就可以换算表达式解决。支持 '/ * + -'四种运算符号, 如: '/100', '*25.2', '+25.3', '-25.6'，如果这几种运算不能满足你的需求，Hw-Link 还提供了自定义数据模型，提供了更复杂的运算模型表达式。后面的章节会讲解。

单位：这个就很好理解了，采集的数据如果有单位显示的需求，可以在这里填写。

是否告警: 如果此数据是告警数据，固定 1 是告警，0 是消除，告警显示信息是告警时 Hw-Link 主动推送的信息，消除信息是当告警消除时的推送消息，如果你不想在消除时推送消息，那么你可以在消除栏里面留空。

是否存储: 如果你想把此条数据存储到数据库中，那么你可以选择是，存储有两种方式，一个是定时存储，一个是变化存储，定时存储是按固定的时间存储，单位是分钟，变化存储是按照设定好的变化值存储，

8.3 SNMP 数据采集

8.3.1 SNMP 设备数据的采集

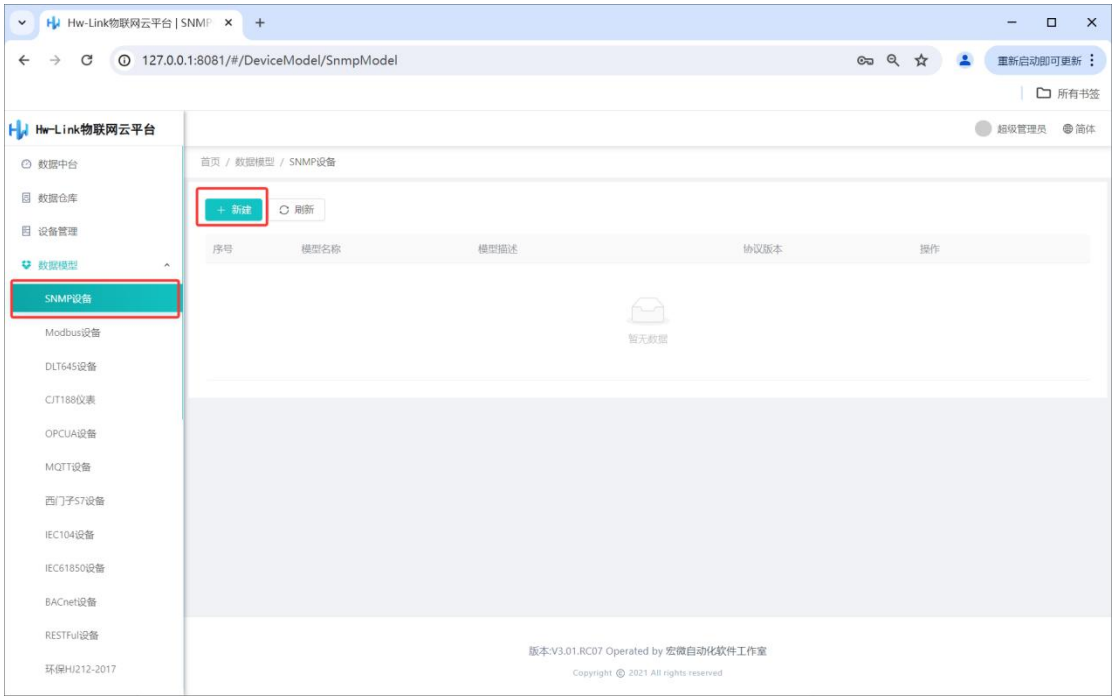
什么是 SNMP 协议

以下是百度百科的介绍

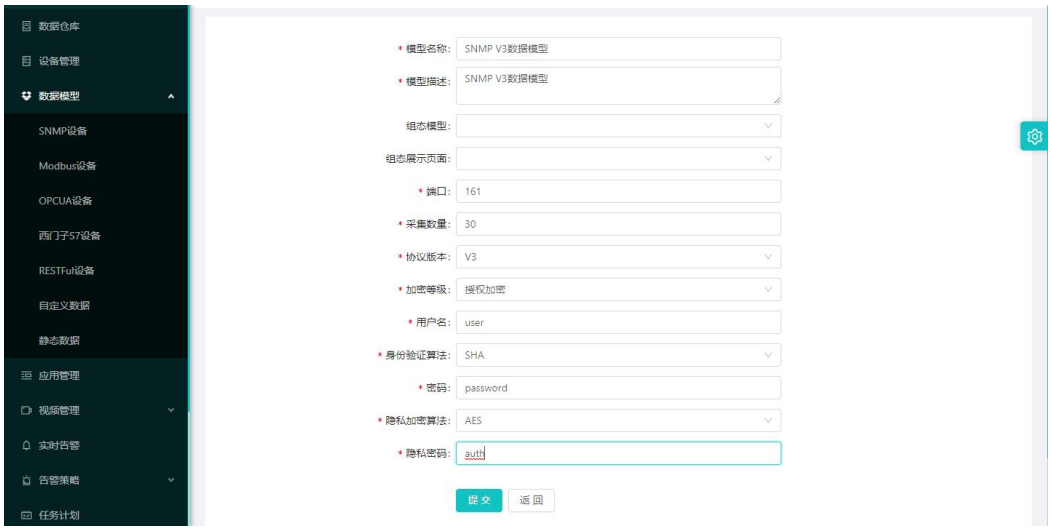
简单网络管理协议(Simple Network Management Protocol--SNMP）的原来名字叫做简单网关监控协议（Simple Gateway Monitoring Protocol-SGMP）。最早是 IETF 的研究小组提出来的，在 SGMP 协议的基础之上，加上新的管理信息结构和管理信息库，让 SGMP 更加全面。简单性和扩展性是 SNMP 所体现出来的，其中包含数据库类型（Database Schema），一个应用层协议（Application Layer Protocol）和一些资料文件。SNMP 管理协议不光能够加强网络管理系统的效能，而且还可以用来对网络中的资源进行管理和实时监控。

创建 SNMP 设备模型

1. 登录 Hw-Link，找到数据模型->SNMP 设备

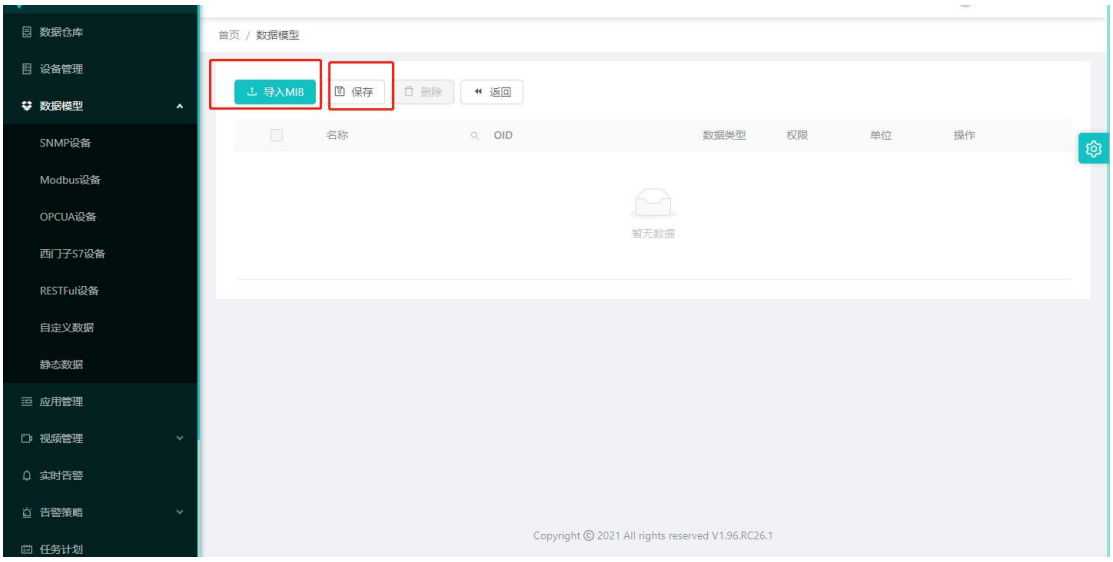


2. 然后点击新建 ,按照界面的提示填写相关的参数，完成后点击提交

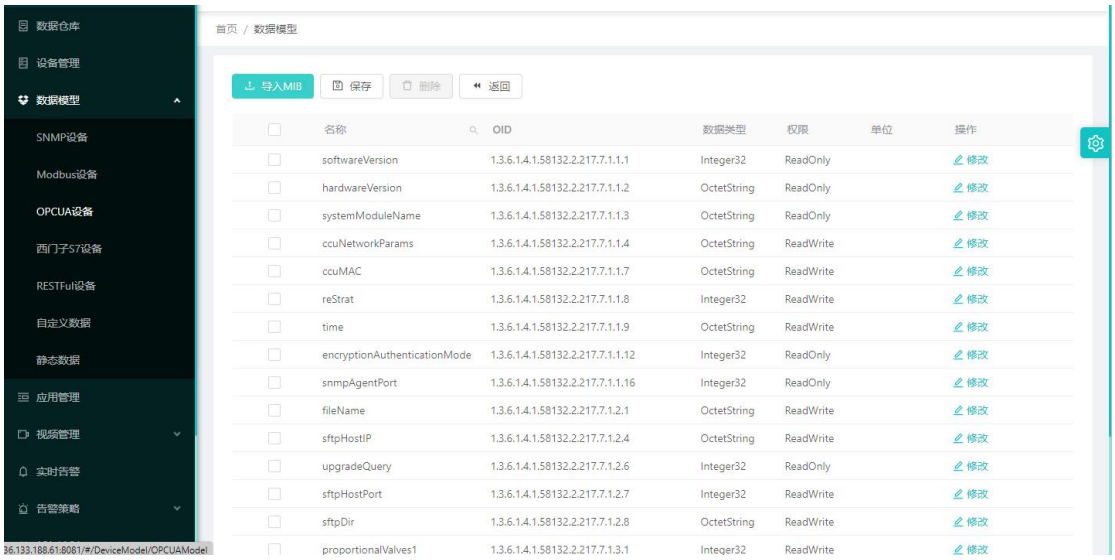


3. 导入 MIB 库文件，找到我们刚才创建的模型，点击导入 MIB

4. 跳转到导入界面，点击导入 MIB 按钮，选择我们设备的 MIB 文件，Hw-Link 会自动解析 MIB 库文件，解析出 OID 列表，完成后要点击保存按钮。

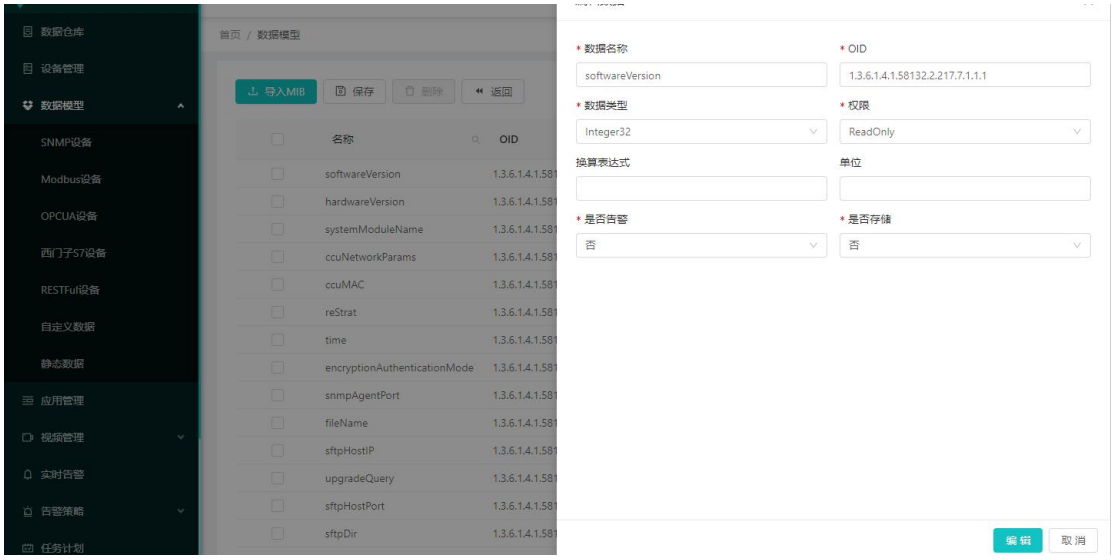


5. Hw-Link 会自动解析 MIB 库文件中的信息，数据类型、权限、OID 路径、名称等



6. 如果想修改数据的单位，换算表达式等信息，点击对应的 OID 数据，然后点击修改

在弹出的界面中可以修改名称、OID、数据类型、权限、换算表达式、单位、是否告警、是否存储等信息。



完成后，下面就可以创建设备，然后采集设备的数据了。

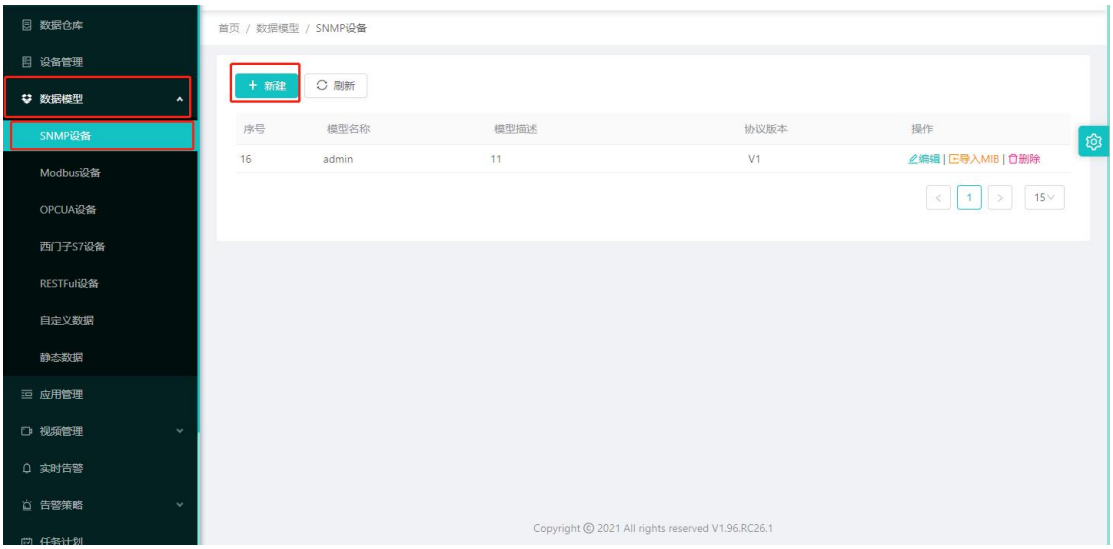
8.3.2 导入 MIB Browser 的 xml 文件

前言

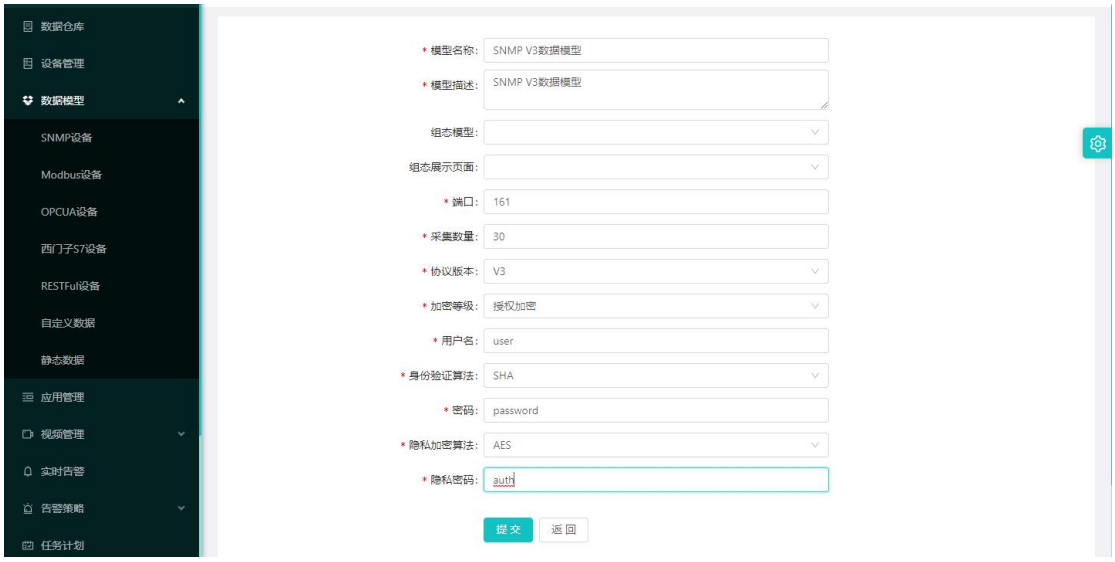
由于 MIB 库文件的复杂性，有些库文件 Hw-Link 可能解析出错，这节特意增加第三方工具解析的文件

创建 SNMP 设备模型

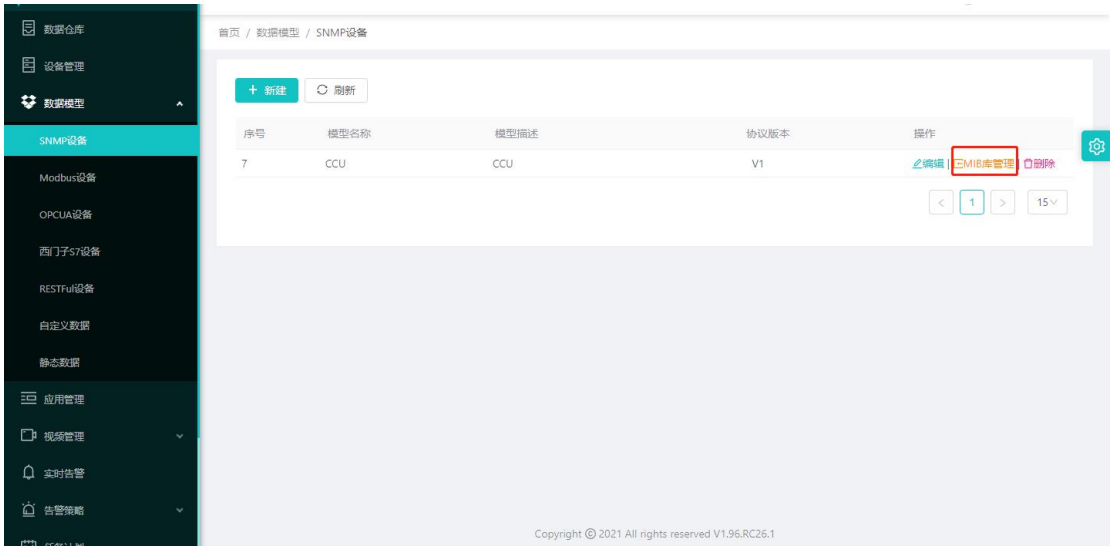
1. 登录 Hw-Link，找到数据模型->SNMP 设备



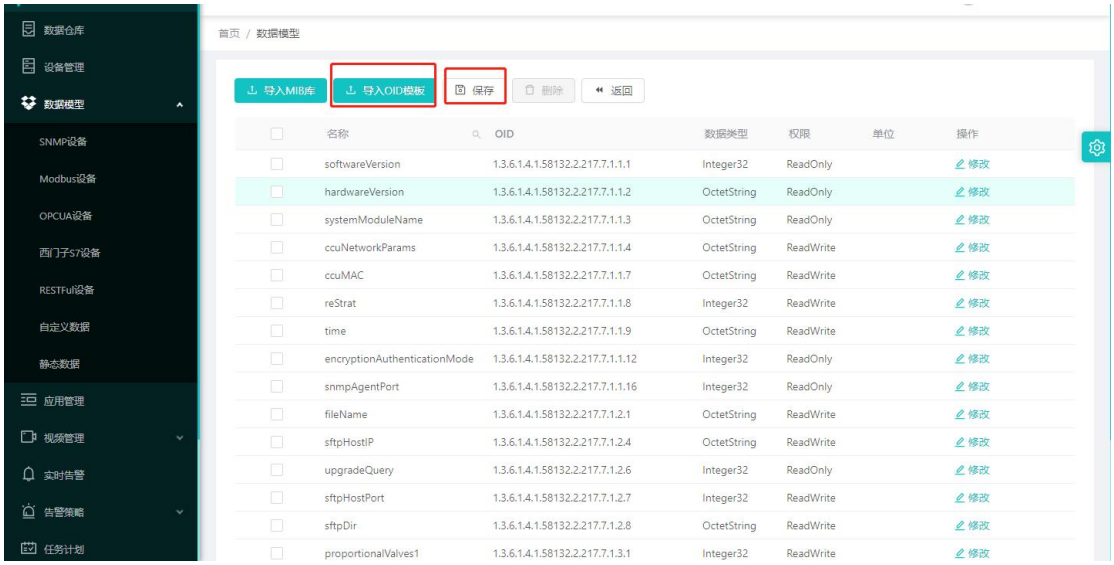
2. 然后点击新建 ,按照界面的提示填写相关的参数，完成后点击提交



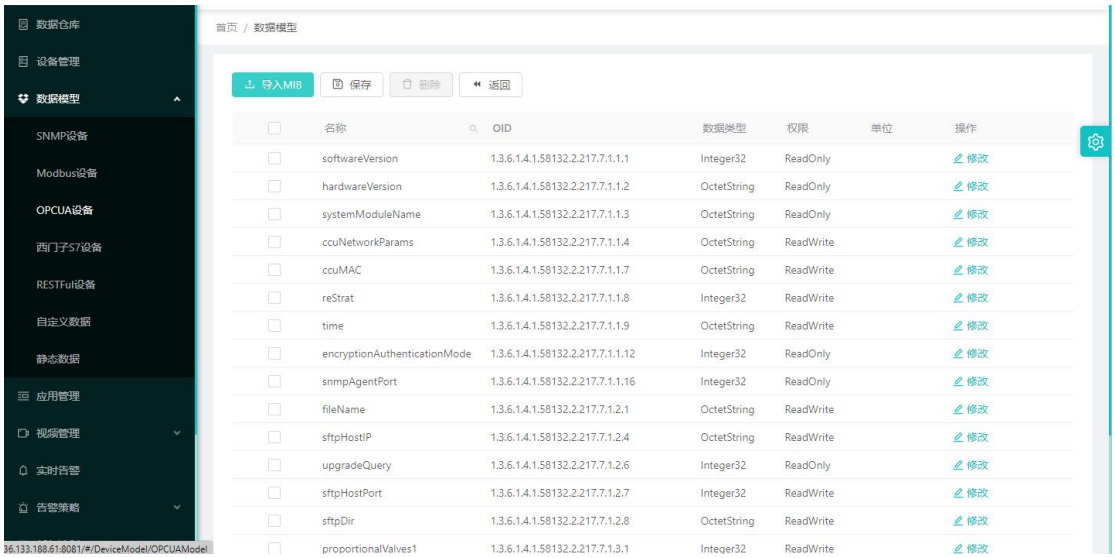
3. 导入 MIB 库文件，找到我们刚才创建的模型，点击 MIB 库管理



4. 跳转到导入界面，点击导入 OID 模板按钮，选择我们刚才导出的 XML

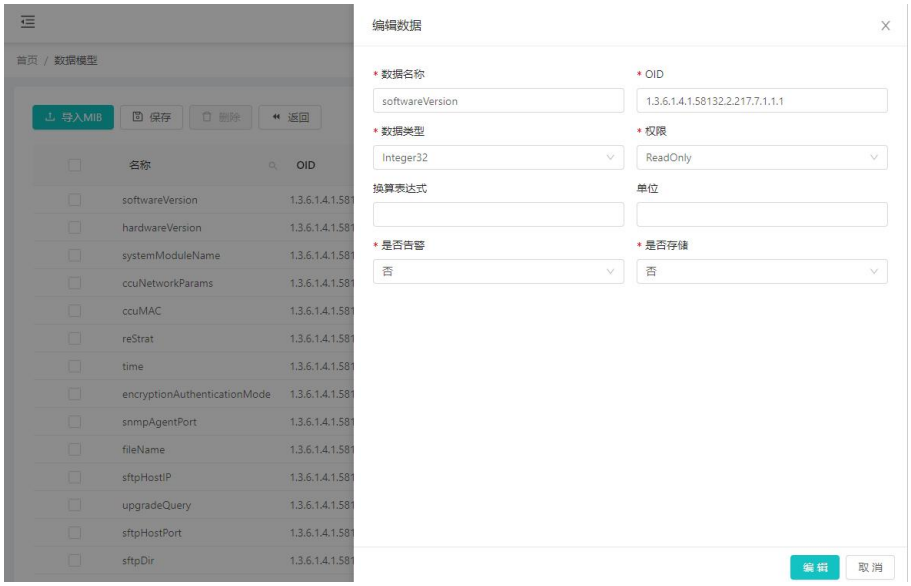


5. Hw-Link 会自动解析模板文件中的信息，数据类型、权限、OID 路径、名称等



6. 如果想修改数据的单位，换算表达式等信息，点击对应的 OID 数据，然后点击修改

在弹出的界面中可以修改名称、OID、数据类型、权限、换算表达式、单位、是否告警、是否存储等信息。



完成后，下面就可以创建设备，然后采集设备的数据了。

8.3.3 SNMP 设备添加和调试

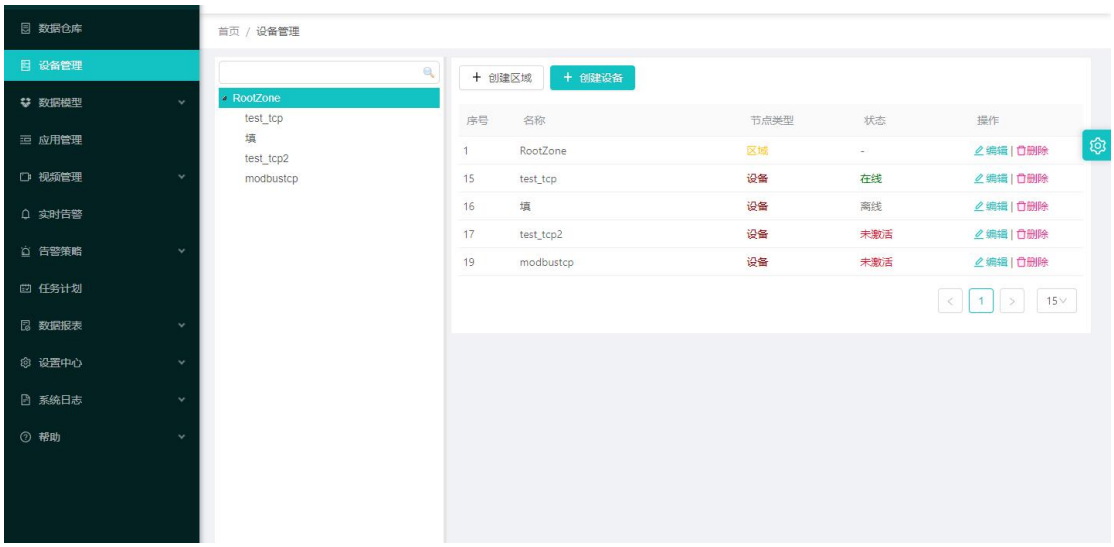
相关信息

上一节我们知道了 SNMP 数据模型的创建，这节我们要根据我们创建的模型，来创建我们准备的设备

8.3.3.1 添加设备

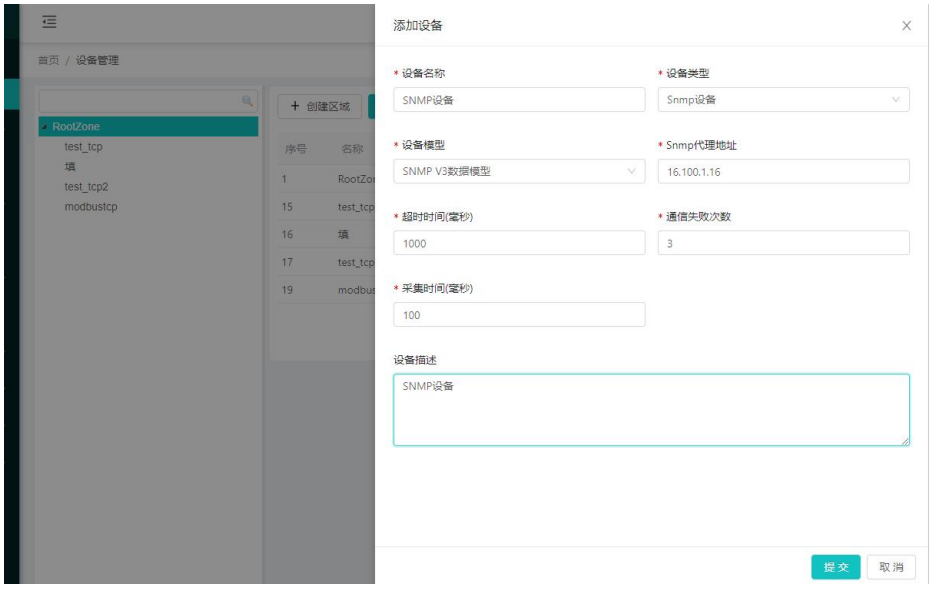
第一步

点击设备管理，然后点击创建设备



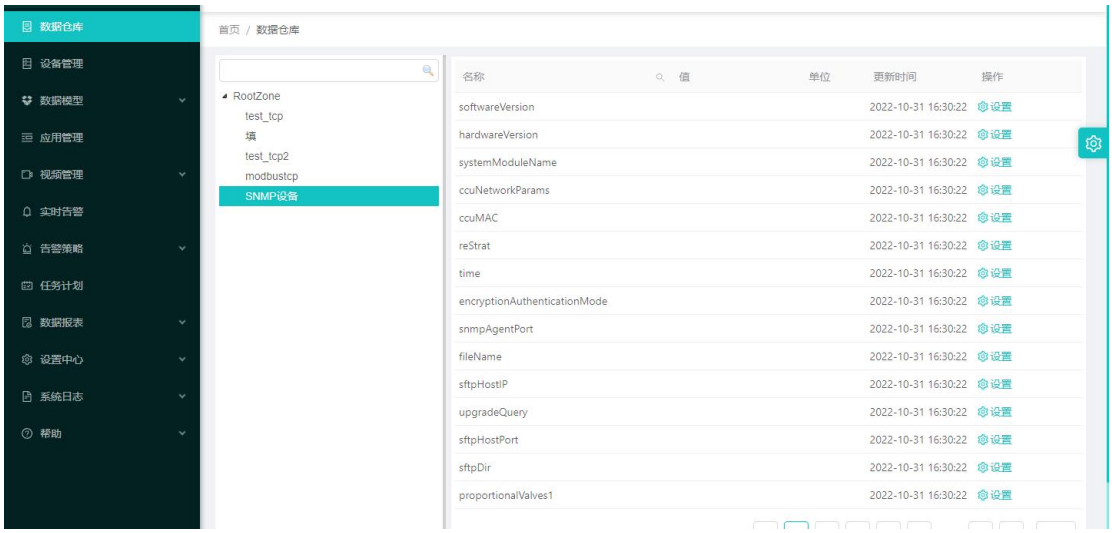
第二步

设备模型我们选择 SNMP 设备，设备模型选择我们刚才创建的 SNMP V3 的模型，代理地址需要根据设备 IP 地址填写。超时、失败次数、采集时间，这三个数据是设备离线判断的标准，(超时时间+采集时间)*次数就是设备离线时间。



8.3.3.2 设备调试

设备添加完后，在数据仓库里面找到我们刚才创建的设备，如果设备网络通畅，这里会实时更新采集的设备数据，如果没有更新，请查看 Hw-Link 控制台的具体错误原因。



8.3.4 利用 MIB Browser 导出 OID 模板文件

前言

由于 MIB 库文件的复杂性，有些库文件 Hw-Link 可能解析出错，这节特意增加第三方工具解析的文件

8.3.4.1.说明

SNMP 客户端工具 MIB Browser，
全名 iReasoning MIB Browser，
是一个功能强大、易于使用的 MIB 管理工具，
支持 Windows,Linux,MACOS 等多种平台。
它通过 SNMP 协议管理网络设备，
它可以加载标准的和私有的 MIB。

8.3.4.2.下载地址

本站下载地址:https://wwt.lanzoum.com/iH4Gb0fm1n8f

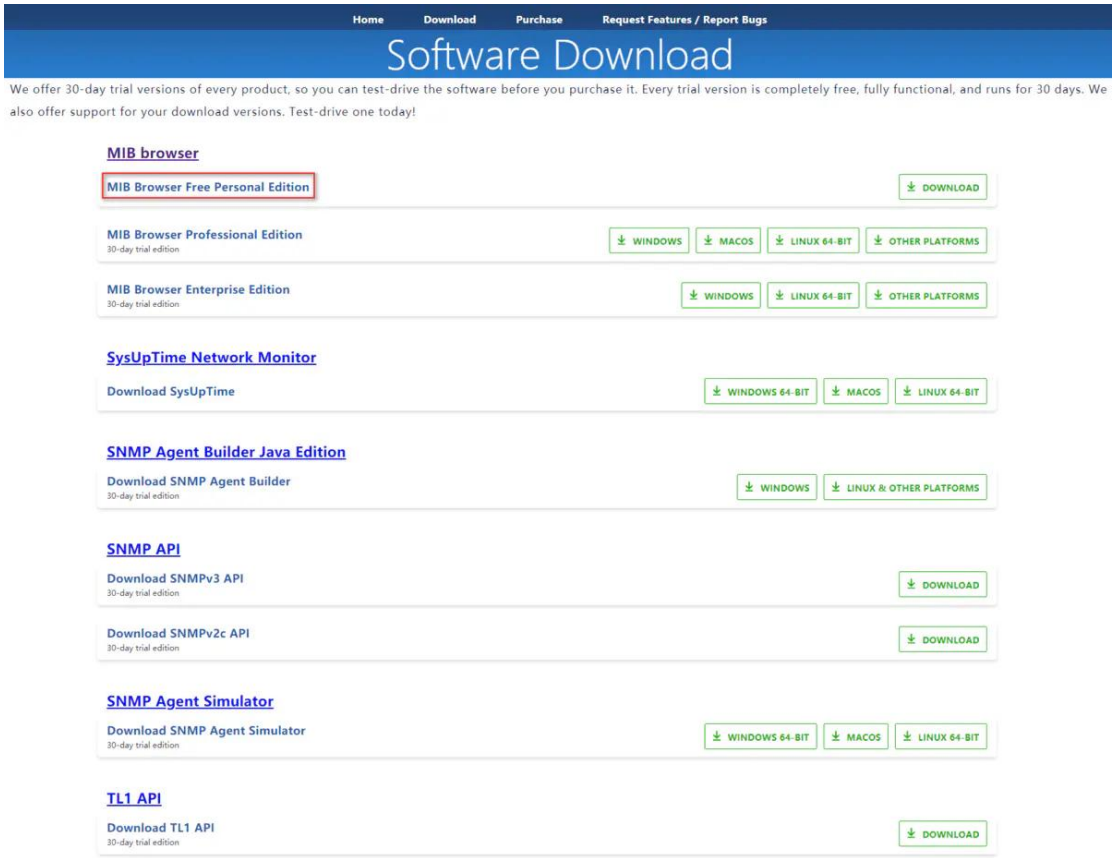
解压后，双击 bin 目录下面的 browser.bat 文件，即可打开 MibBrowser 。

官方下载地址：

http://www.ireasoning.com/download.shtml

提供个人免费版本下载：

MIB Browser Free Personal Edition



8.3.4.3.加载 MIB

依次点击 File -> Load MIBs,

可以加载指定的 MIB 文件，

然后通过左边的 MIB Tree 视图，

点击对应的 MIB 属性，

能够自动填充选中属性的 OID 值，

方便进行快速查询和设置

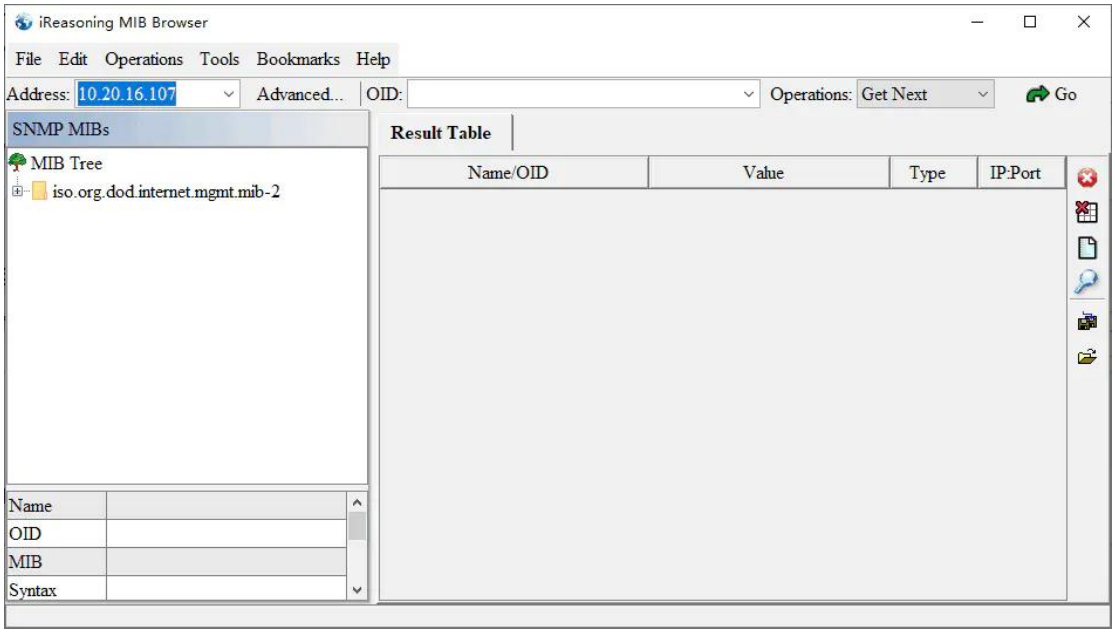
8.3.4.4.连接设备

打开 MIB Browser 软件后，

在 Address 中输入设备 IP 地址，

需要确保设备开启了 SNMP 协议，

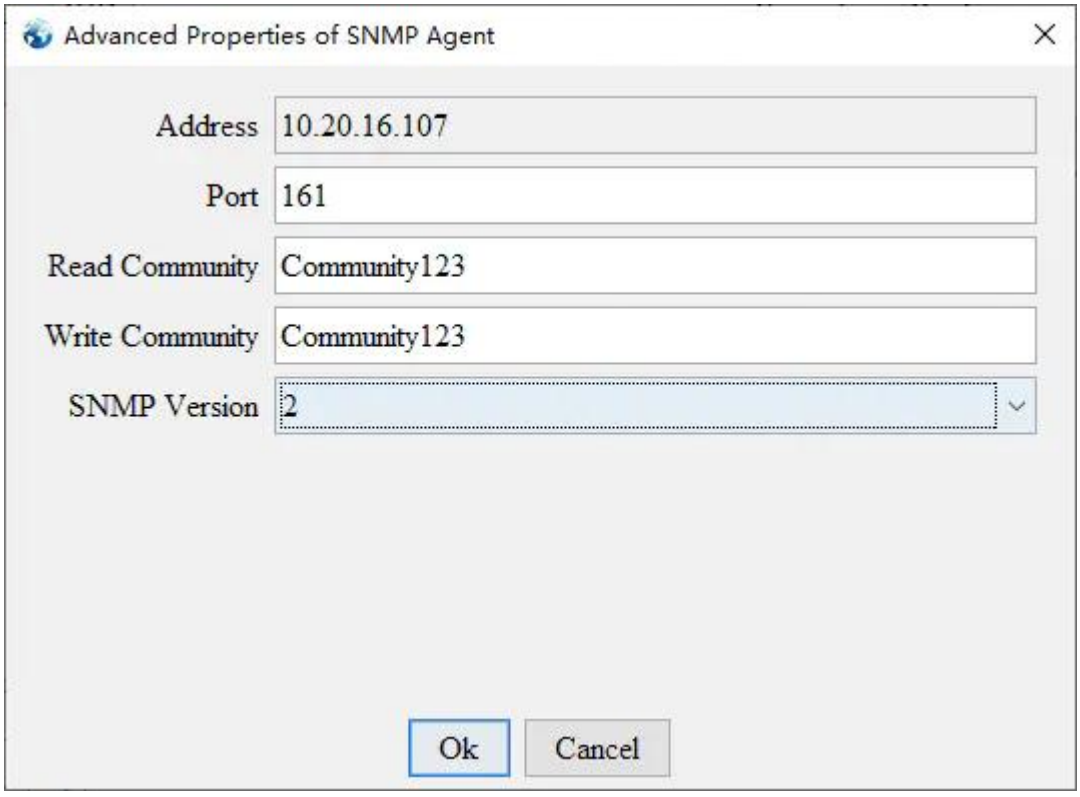
默认情况下 Port 端口是 161：



如果需要进一步修改连接配置，

点击 Address 右边的 Advanced...，

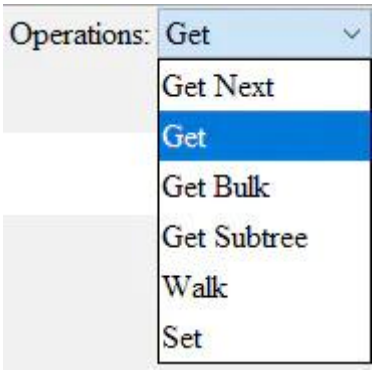
可以修改 Port，Community，SNMP 版本等属性:



8.3.4.5.操作类型

工具支持如下 6 种 Operations 操作类型，

这里简单说明一下：



1. Get 获取当前 OID 的值
2. Get Next 获取下一个 OID 的值，会自动增加 OID 序号
3. Get Bulk 分页批量获取，一次获取 10 个值，会自动增加 OID 序号
4. Get Subtree 获取 OID 下面的所有子树的值
5. Walk 获取 OID 的值，会自动增加 OID 序号，且一直持续查询
6. Set 设置 OID 的值

8.3.4.6.查询数据

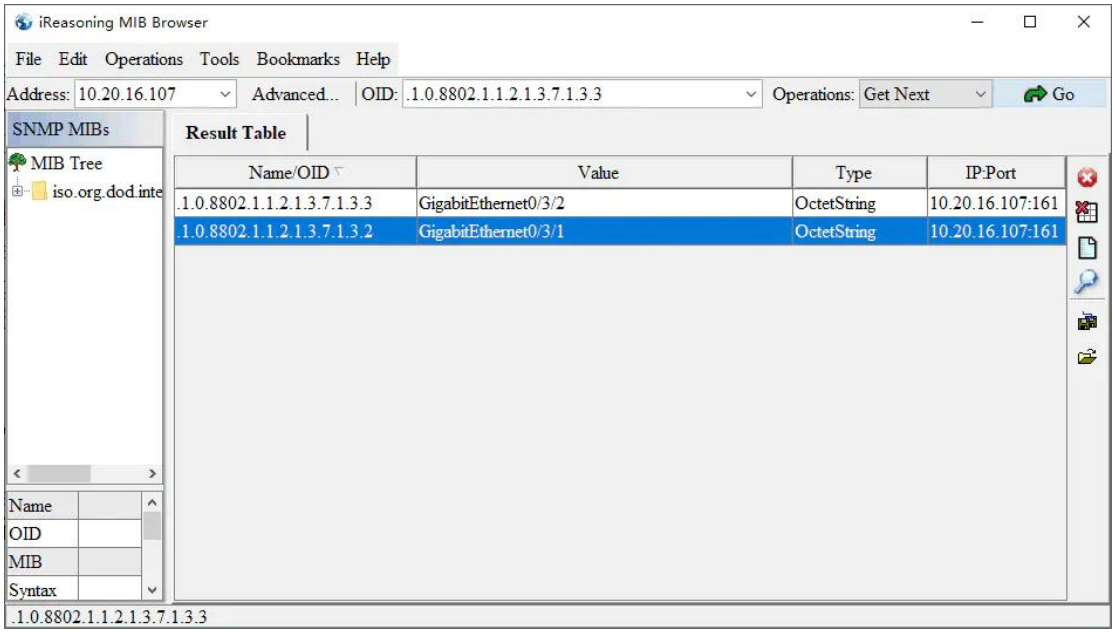
通过输入 MIB 对应的 OID，

可以查询对应的数据。

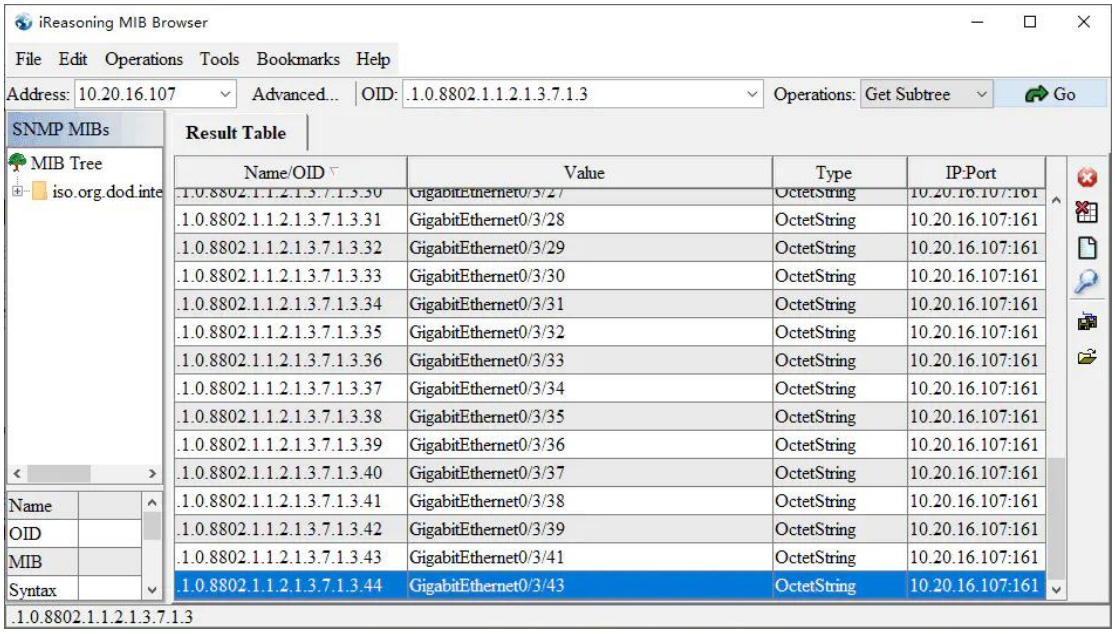
选择对应的操作类型 Operations,

点击最右边的 Go 即可查询数据，

Operations 选择 Get Next：



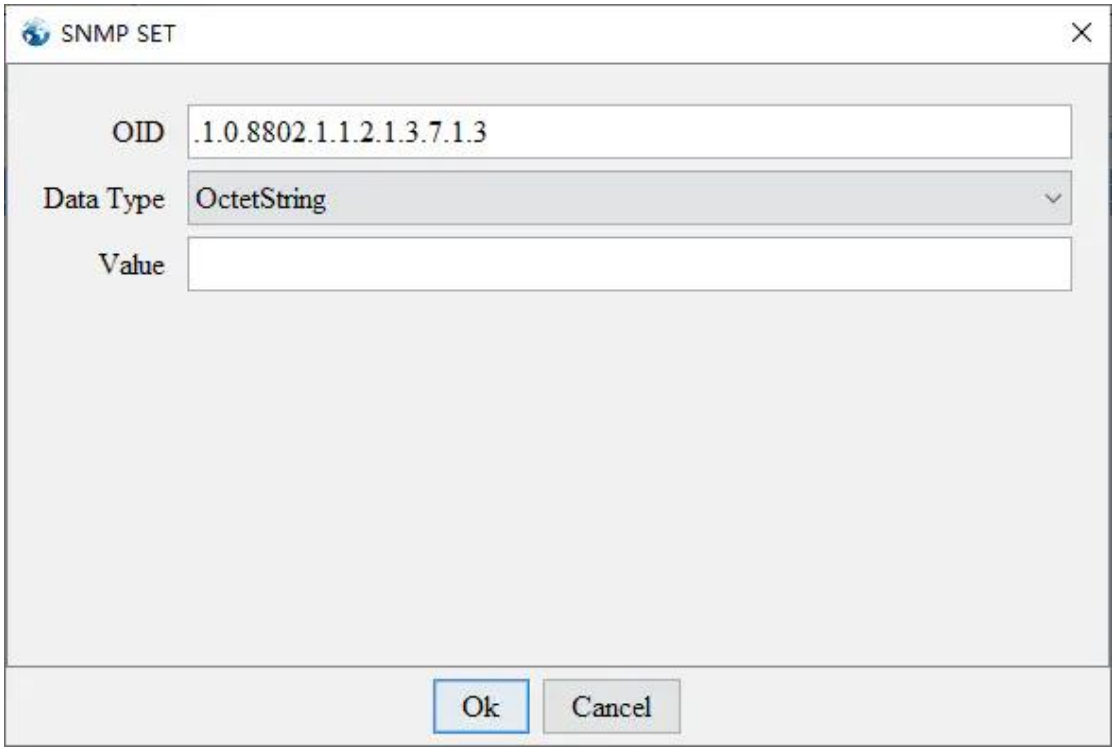
Operations 选择 Get Subtree：



8.3.4.7.设置数据

通过 Set 操作类型，

可以设置 OID 对应的值：



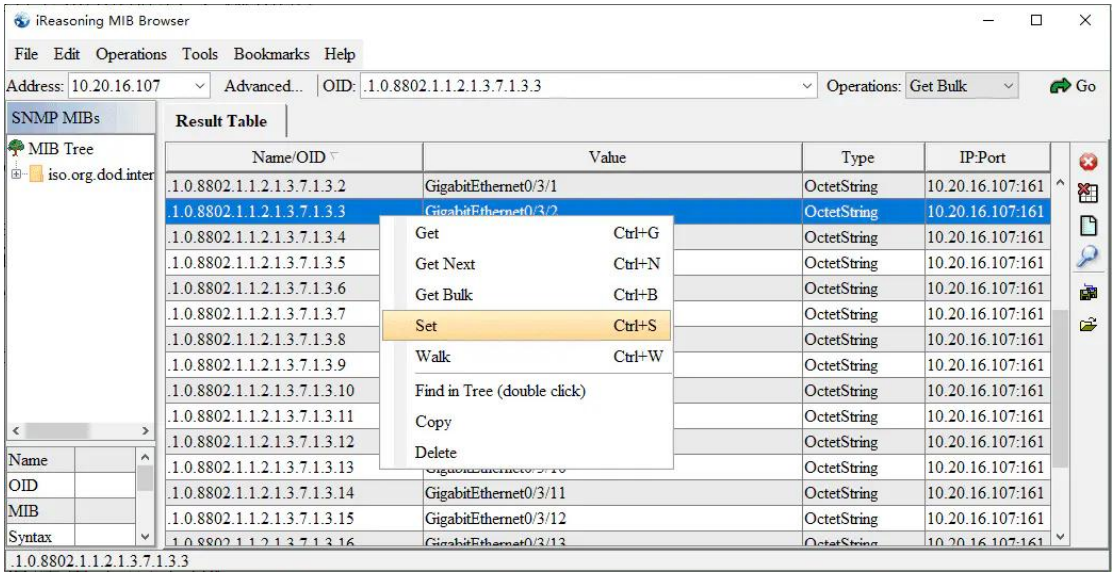
也可以选中查询出来的结果，

在这条记录上面右键，

点击 Set，

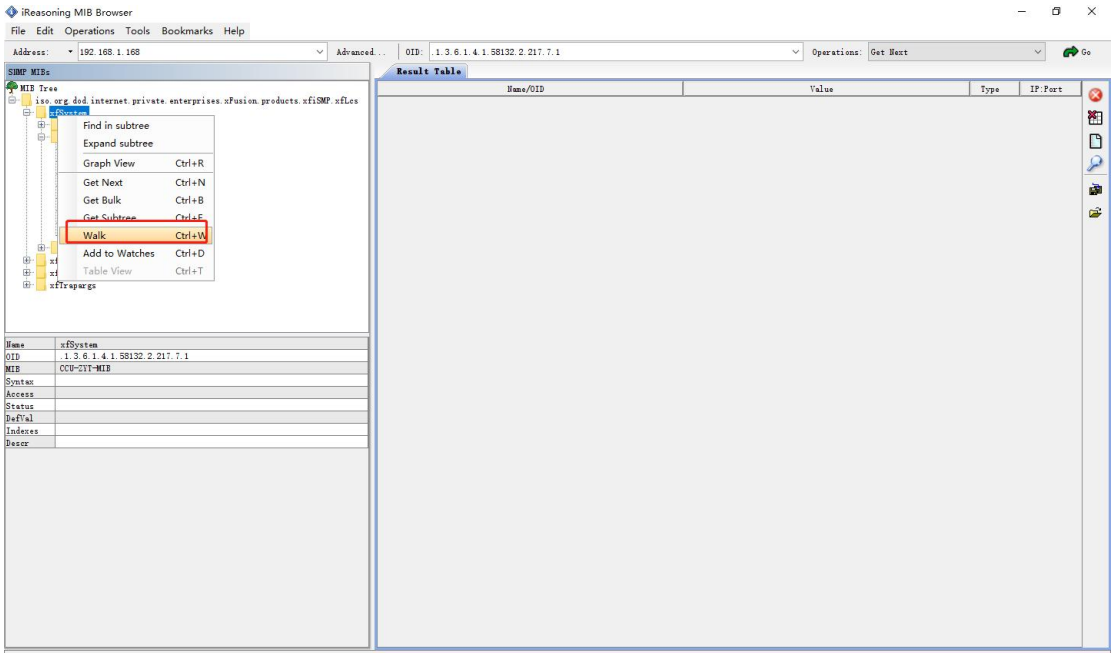
或者快捷键 Ctrl+S，

同样可以设置 OID 对应的值:

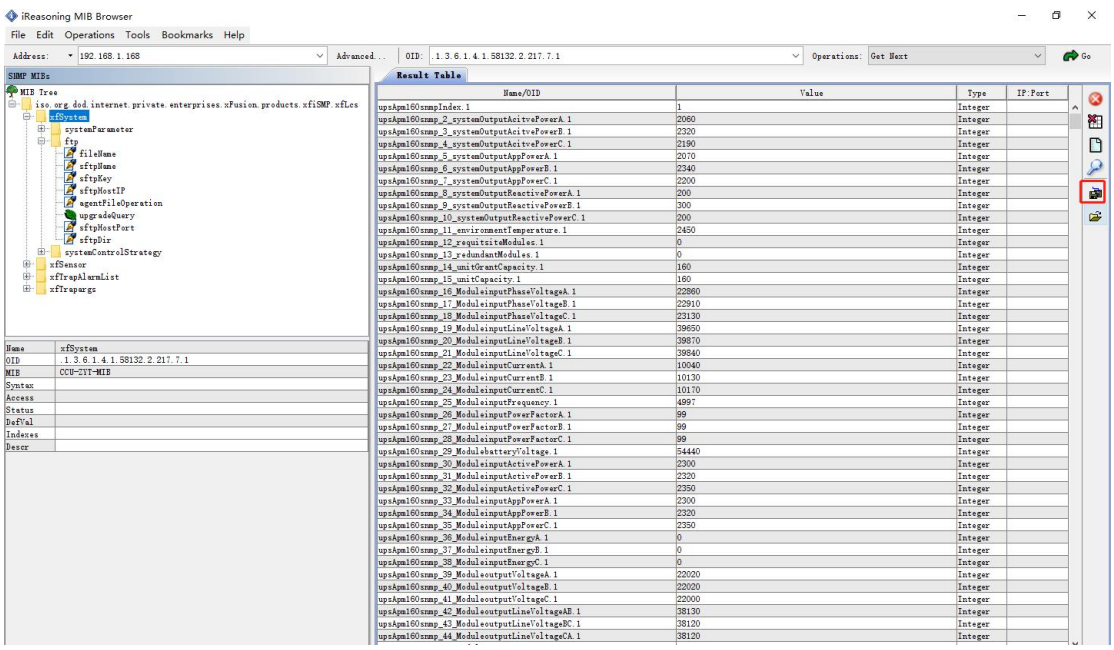


8.3.4.8.导出 XML 文件模板

右击 MIB 的第一个节点，选择 Walk



如果设备通信正常，在右边的栏中会列出设备中所有的 OID 信息，等采集完毕，我们点击保存按钮，即可把所有的 OID 信息保存到模板文件中



8.4 Modbus 数据采集

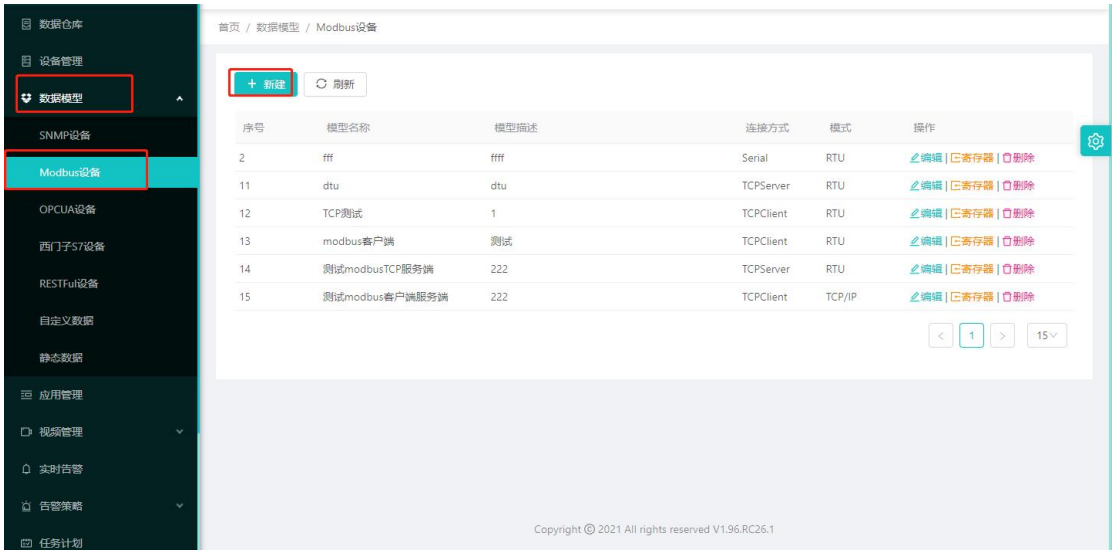
8.4.1 创建 Modbus 数据模型

相关信息

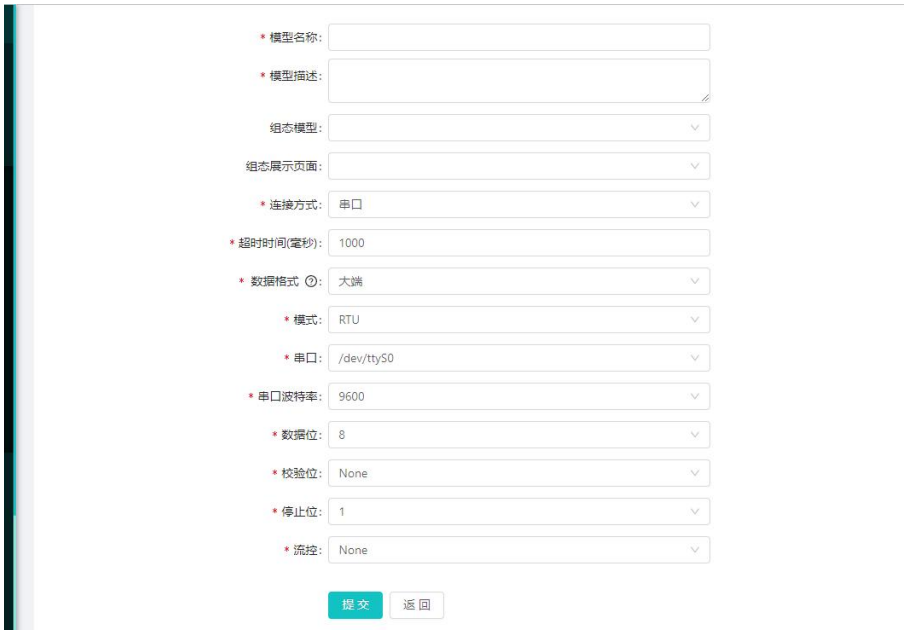
本节讲述 Modbus 寄存器的创建

8.4.1.1 步骤 1

登录 Hw-Link，依次点击数据模型->Modbus 设备->新建



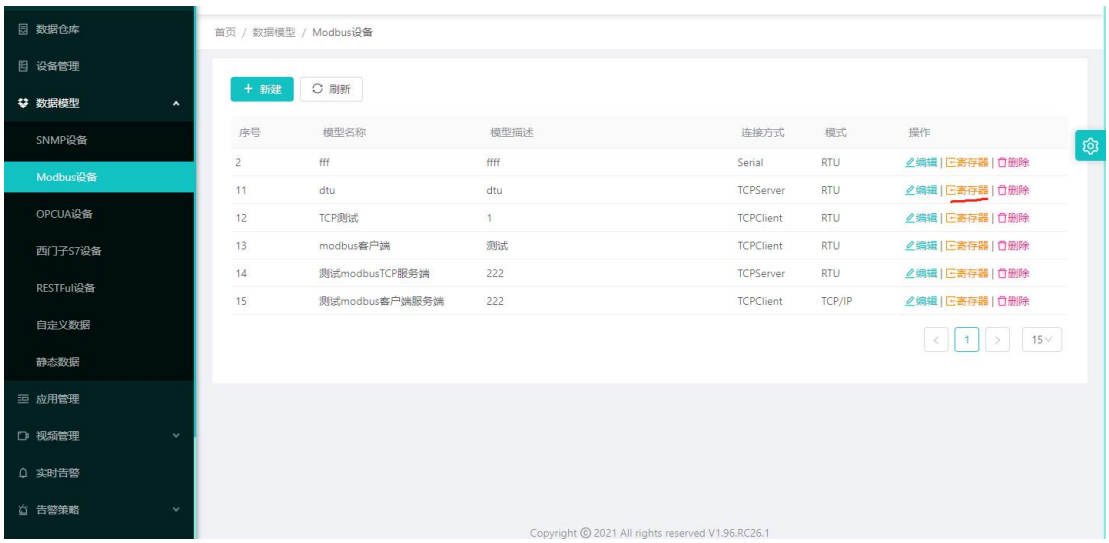
根据页面的提示填写好 Modbus 的通信参数，点击保存，目前支持 RTU、ASCII、TCP、TCP Server 通信方式。根据自己的需要自行选择添加。



8.4.1.2 步骤 2

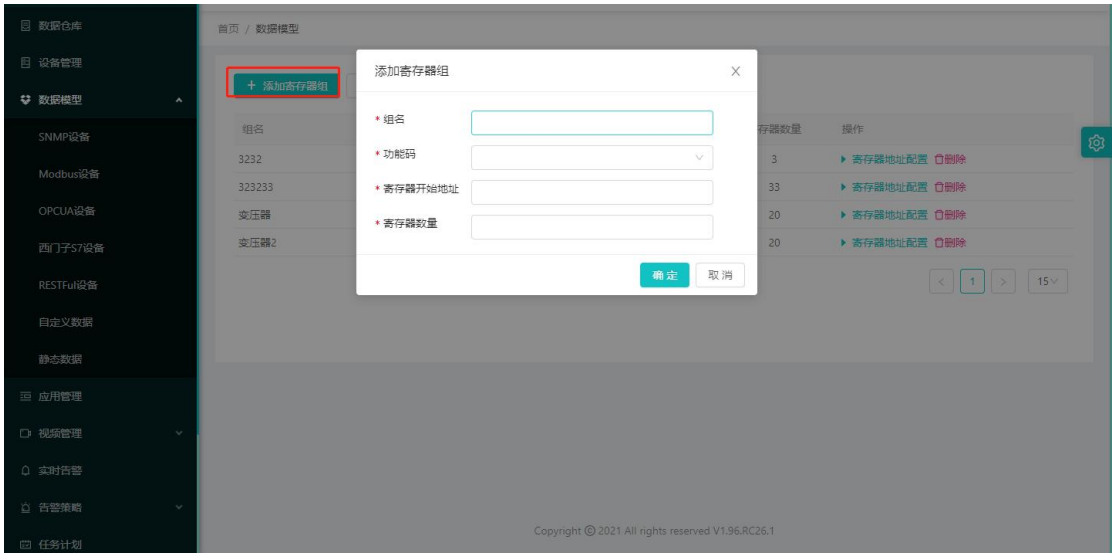
通信模型创建后，我们要创建 Modbus 的寄存器数据，Hw-Link 支持多组寄存器的读写，支持 01、02、03、04、06 功能码

找到刚才我们创建的 Modbus 模型，点击寄存器，跳转到模型的寄存器配置页面



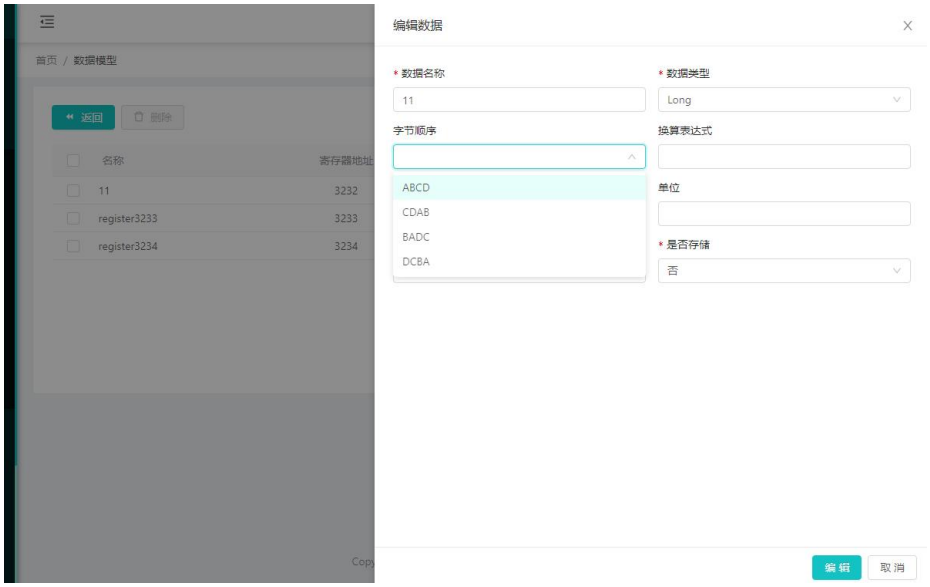
点击添加寄存器组，就可以创建寄存器，开始地址是此寄存器组的 Modbus 寄存器开始读取地址，数量是此寄存器连续读取多少个寄存器，填写完成后，Hw-Link 会自动根据开始寄存器和数量，自动创建寄存器数据。

注意：此处的开始地址不能带功能码。比如要使用 03 号功能码采集 400100 开始的保持寄存区，那么此处的开始地址需要填写 100，不能填写 400100.其它功能码类此。



如果想修改自动创建的每个寄存器的名称、数据类型、是否告警、是否存储等数据属性，点击寄存器地址配置，可以修改每个寄存器的属性，注意，Hw-Link 已经可以支持 modbus 的长整型和浮点型的读取。

长整型和浮点型，Hw-Link 支持四种字节顺序的读取，根据设备的数据说明选择相应的字节顺序。



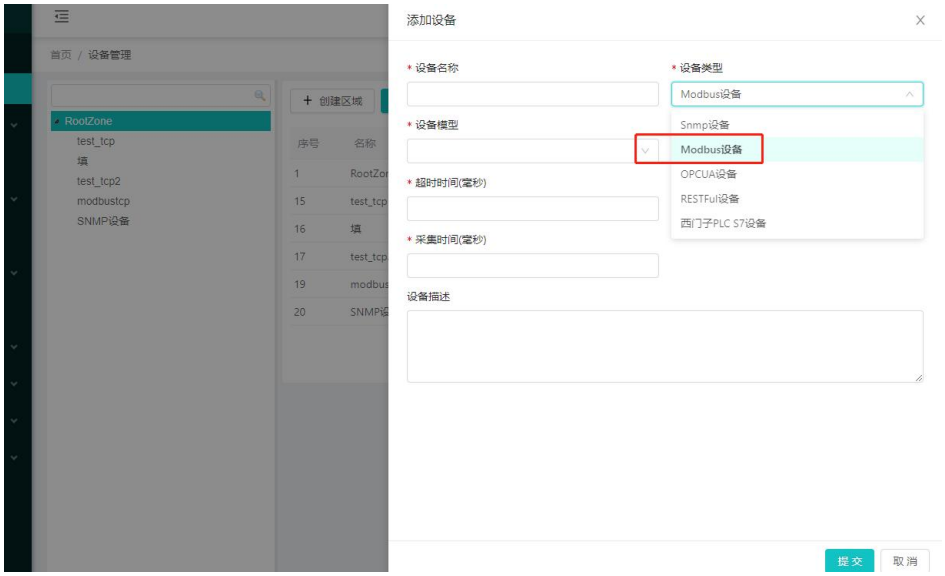
8.4.2 创建 Modbus 设备

相关信息

本节讲述 modbus 设备的创建

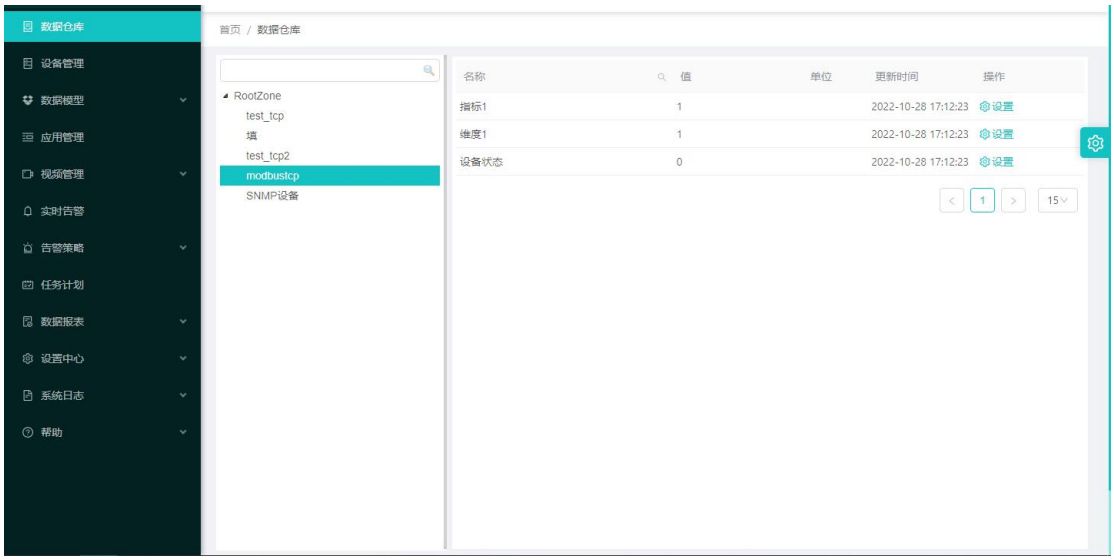
8.4.2.1 步骤 1

点击设备管理，点击创建设备，弹出的页面中设备类型选择 Modbus 设备，设备模型选择我们刚才创建好的数据模型。



8.4.2.2 步骤 2

设备创建完成后，点击数据仓库，即可查看设备采集的数据



8.4.3 通过 DTU 连接 Hw-Link，并采集 Modbus 设备数据

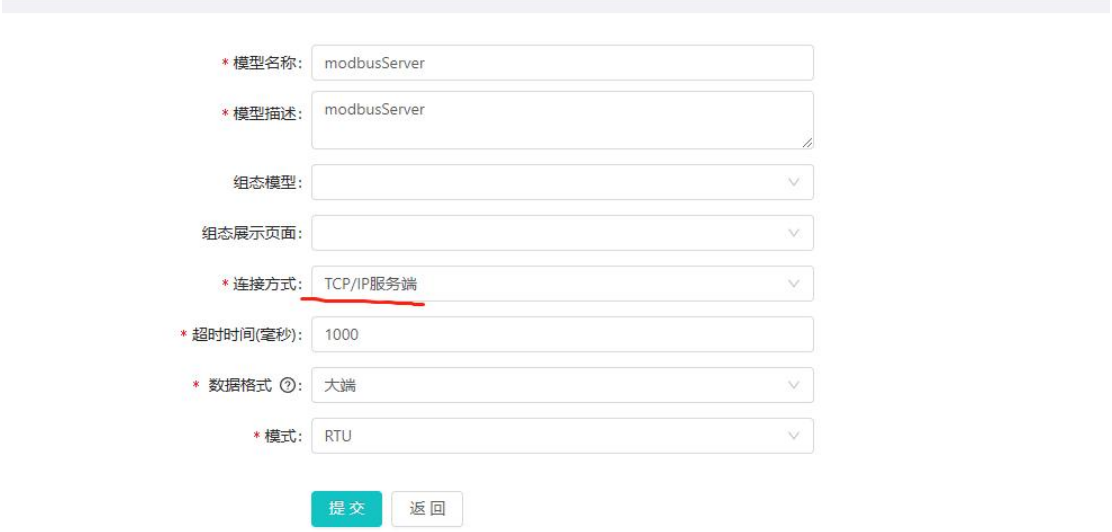
相关信息

本节讲述通过 DTU 的透传方式接入 Hw-Link，并采集 DTU 后端的 Modbus 设备数据。注意：Hw-Link 开放 3000 端口供 DTU 连接。

8.4.3.1 创建数据模型

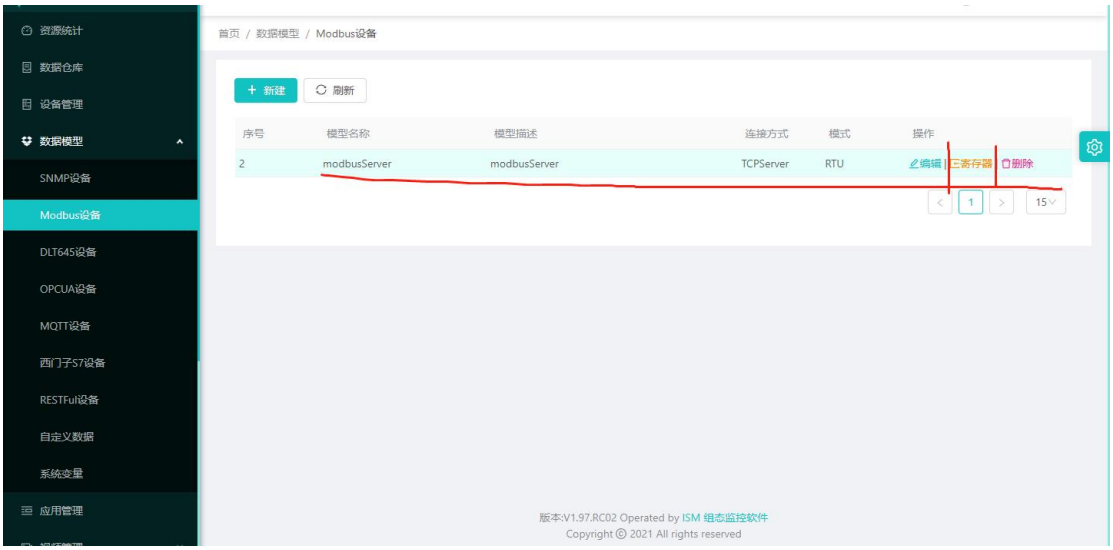
步骤 1

创建 modbus server 数据模型。



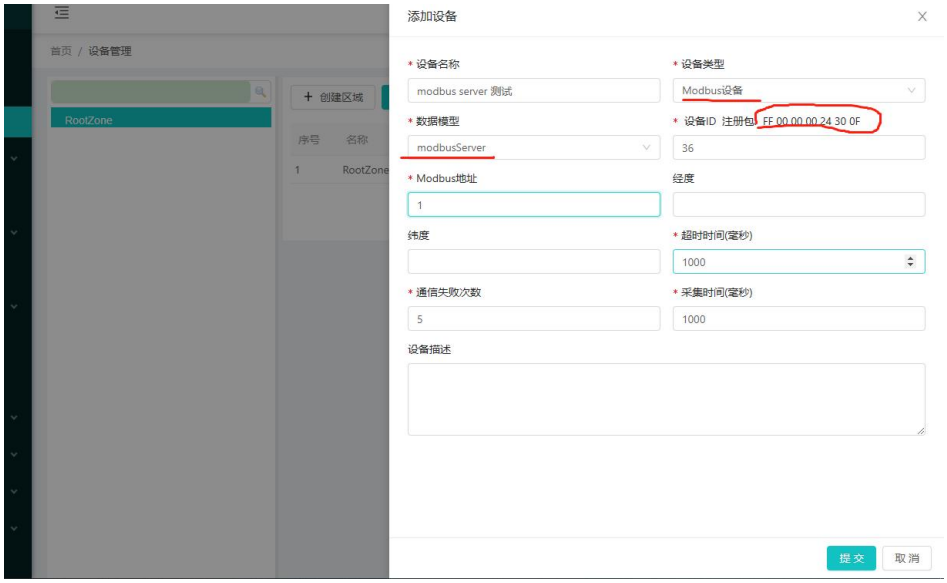
步骤 2

创建寄存器点表



步骤 3

创建设备,注册包是 DTU 连接后必须要先发送的数据包，用于标识连接的通道信息。DTU 要以十六进制发送，模式是连接后发送。
DTU 不需要发送心跳包。



8.5 DLT645 协议采集

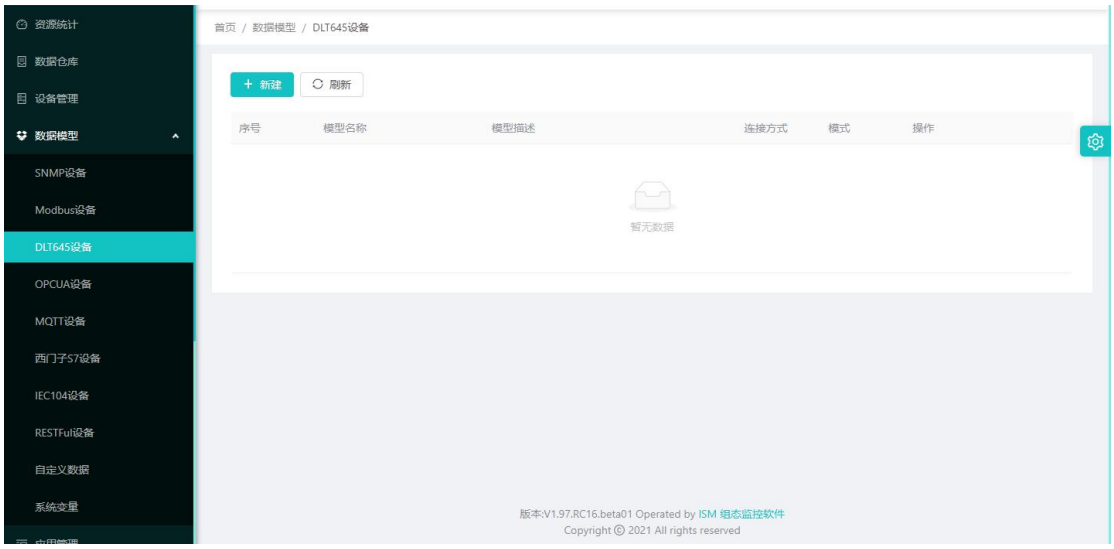
8.5.1 创建 DLT645 数据模型

相关信息

DLT645 数据驱动只支持 2007 规约

8.5.1.1 步骤 1

登录 Hw-Link，依次点击数据模型->DLT645 设备->新建



8.5.1.2 步骤 2

根据页面的提示填写好 DLT645 的通信参数，点击保存，目前支持 串口、TCP 客户端、TCP Server 通信方式。根据自己的需要自行选择添加。

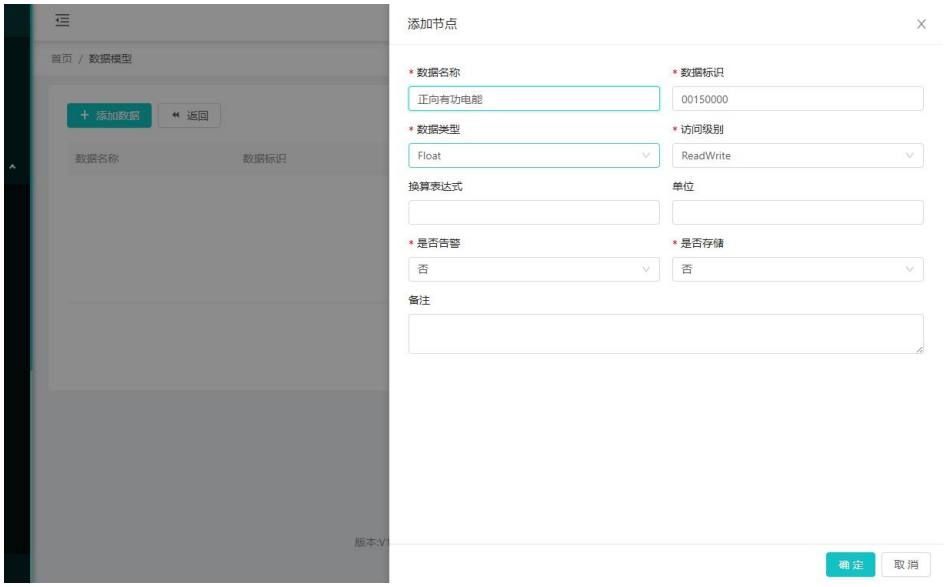
8.5.2 添加 DLT645 数据点

8.5.2.1 步骤 1

登录 Hw-Link，选择我们刚才新建的 DLT645 的数据模型

8.5.2.2 步骤 2

根据页面的提示填写好 DLT645 的数据信息,数据标识是 DLT645 协议规范里面的数据标识。然后把采集的数据一一填写好



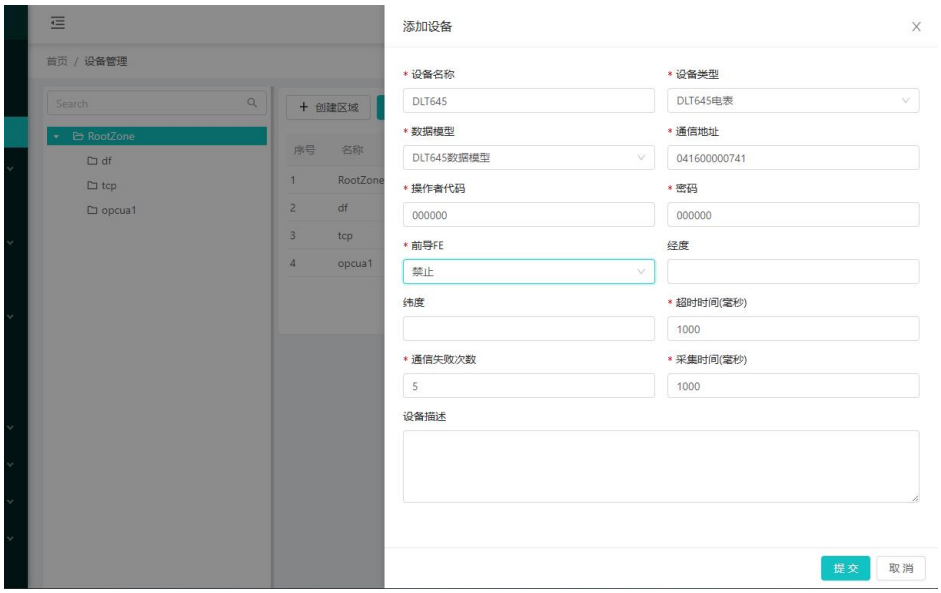
8.5.3 创建 DLT645 设备

相关信息

本节讲述 DLT645 设备的创建

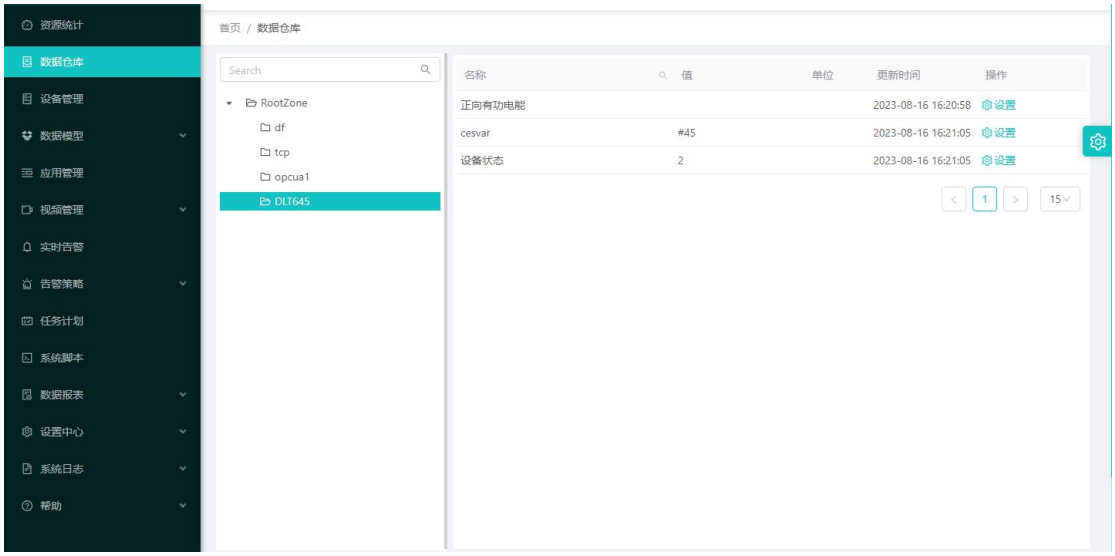
8.5.3.1 步骤 1

点击设备管理，点击创建设备，弹出的页面中设备类型选择 DLT645 设备，设备模型选择我们刚才创建好的数据模型。



8.5.3.2 步骤 2

设备创建完成后，点击数据仓库，即可查看设备采集的数据



8.5.4 通过 DTU 连接 Hw-Link，并采集 DLT645 数据

相关信息

本节讲述通过 DTU 的透传方式接入 Hw-Link，并采集 DTU 后端的 DLT645 设备数据。注意：Hw-Link 开放 10216 端口供 DTU 连接。

8.5.4.1 步骤 1

创建 DLT645 server 数据模型。

8.5.4.2 步骤 2

创建设备,注册包是 DTU 连接后必须要先发送的数据包，用于标识连接的通道信息。DTU 要以十六进制发送，模式是连接后发送。

DTU 不需要发送心跳包。

8.6 IEC 协议采集

8.6.1 创建 IEC104 数据模型

相关信息

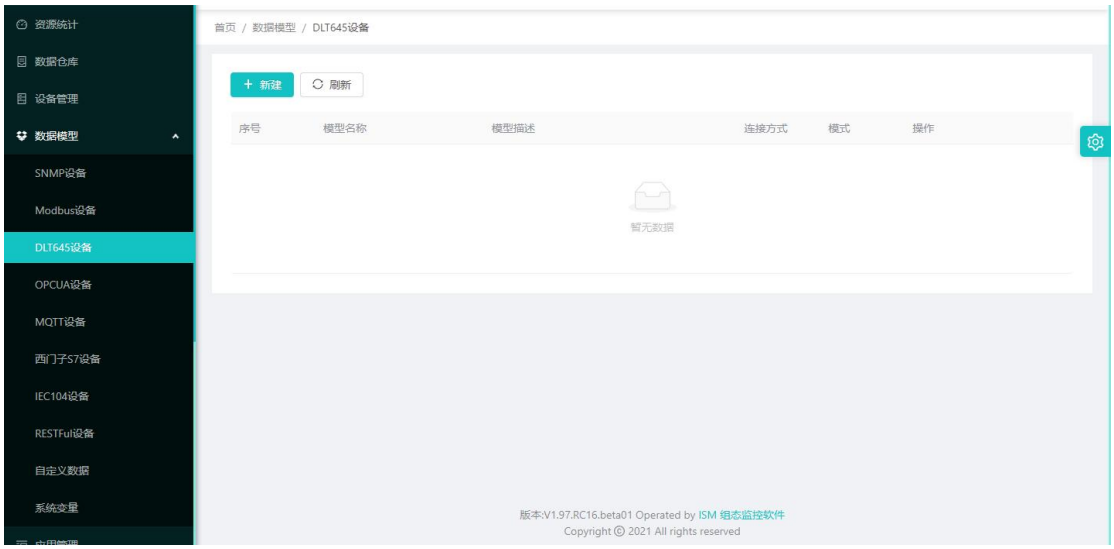
IEC104 规约是一个广泛应用于电力、城市轨道交通等行业的国际标准。

快速了解电力 IEC104 协议规约

https://blog.csdn.net/wgd0707/article/details/122344581

8.6.1.1 步骤 1

登录 Hw-Link，依次点击数据模型->IEC104 设备->新建



8.6.1.2 步骤 2

根据页面的提示填写好 IEC104 的模型参数，点击提交

8.6.2 添加 IEC104 数据点

相关信息

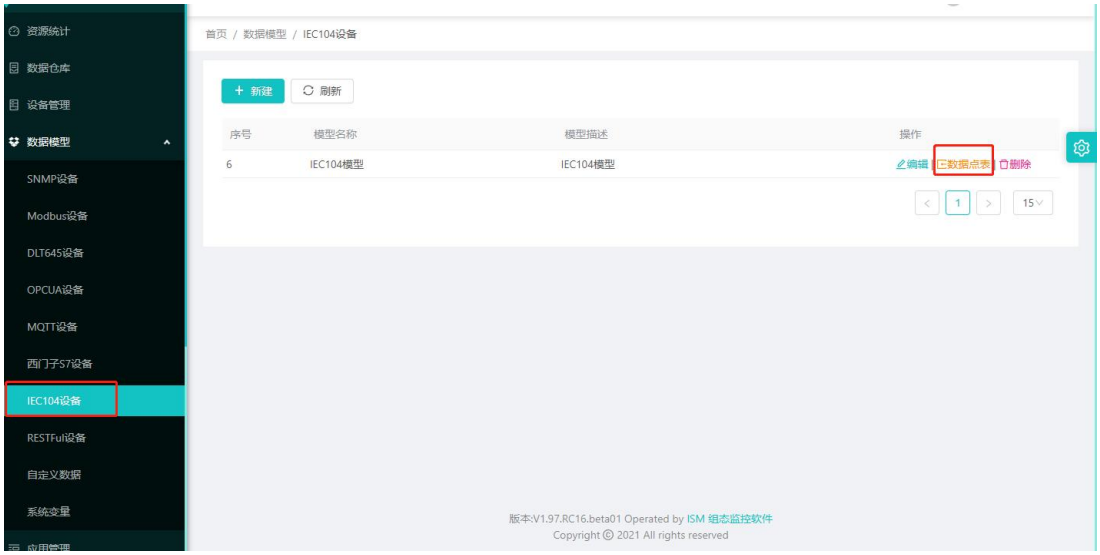
IEC104 规约是一个广泛应用于电力、城市轨道交通等行业的国际标准。

快速了解电力 IEC104 协议规约

https://blog.csdn.net/wgd0707/article/details/122344581

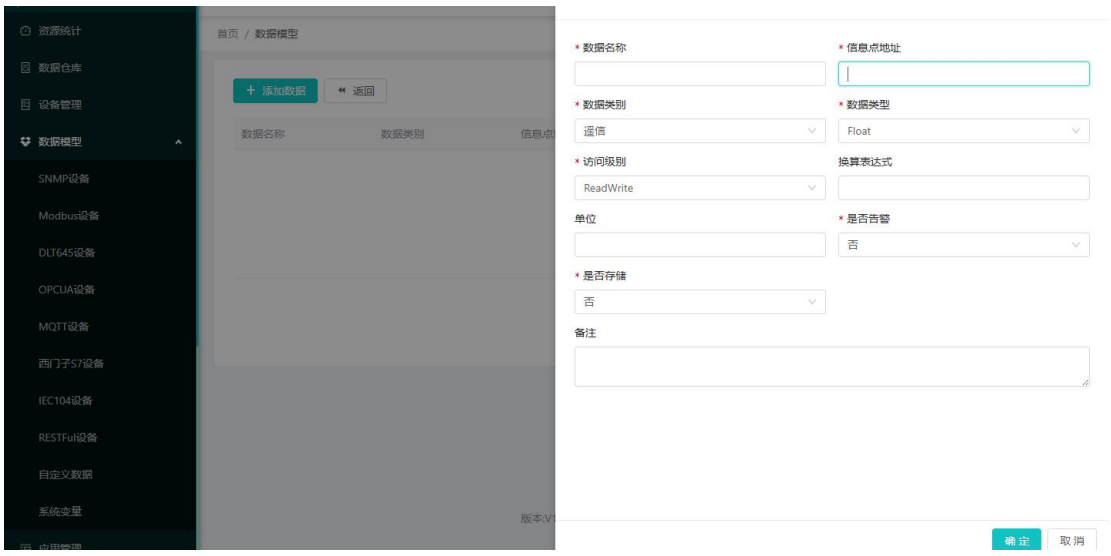
8.6.2.1 步骤 1

登录 Hw-Link，选择我们刚才新建的 IEC104 的数据模型



8.6.2.2 步骤 2

根据页面的提示填写好 IEC104 的数据信息,信息点地址是 IEC104 协议规范里面的信息点。然后把采集的数据一一填写好



8.6.3 创建 IEC104 设备

相关信息

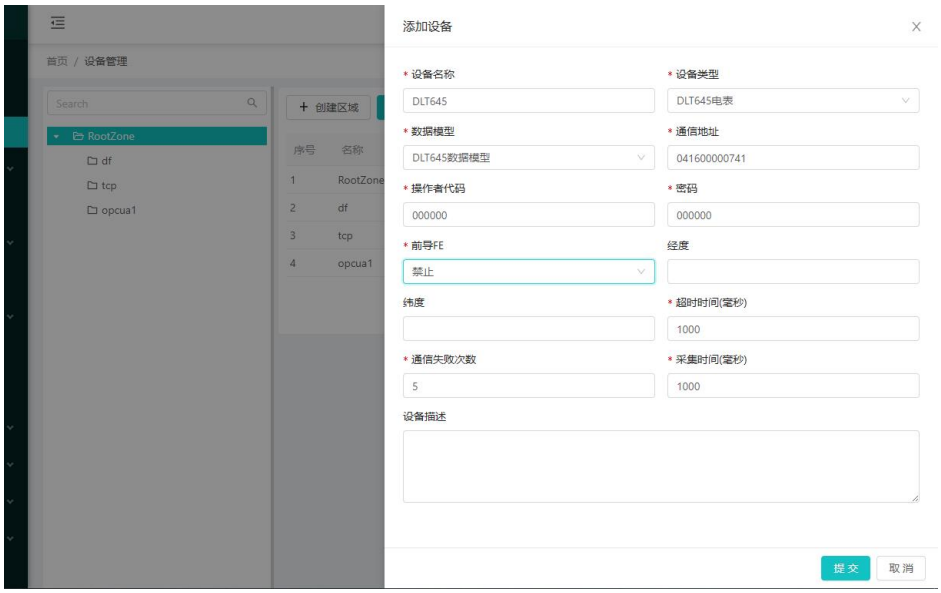
IEC104 规约是一个广泛应用于电力、城市轨道交通等行业的国际标准。

快速了解电力 IEC104 协议规约

https://blog.csdn.net/wgd0707/article/details/122344581

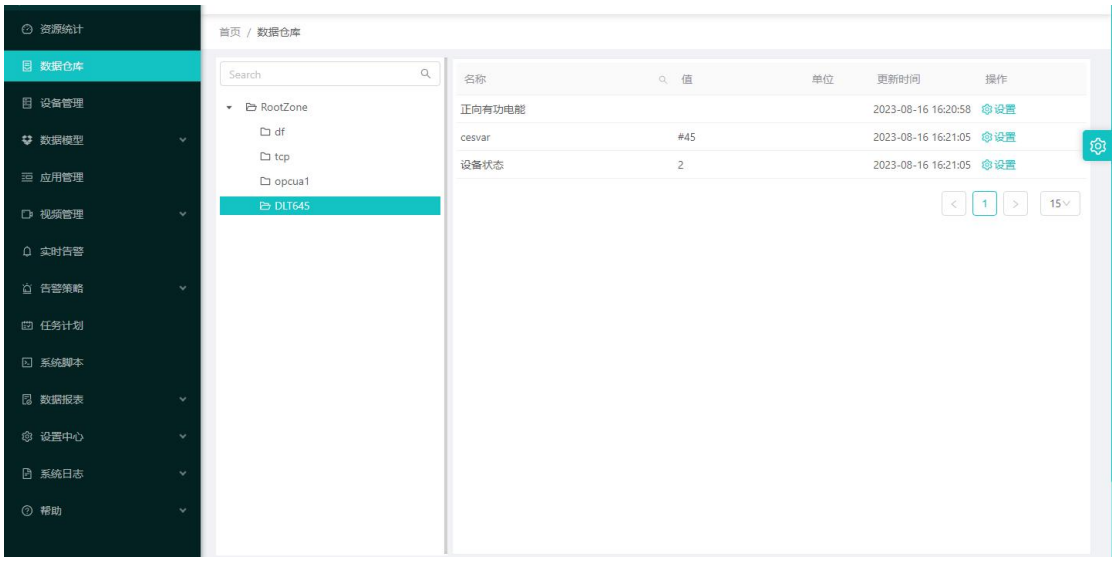
8.6.3.1 步骤 1

点击设备管理，点击创建设备，弹出的页面中设备类型选择 DLT645 设备，设备模型选择我们刚才创建好的数据模型。



8.6.3.2 步骤 2

设备创建完成后，点击数据仓库，即可查看设备采集的数据



8.7 OPC UA 协议采集

8.7.1 创建 OPC UA 的数据模型

本节讲述 OPC UA 数据模型的创建

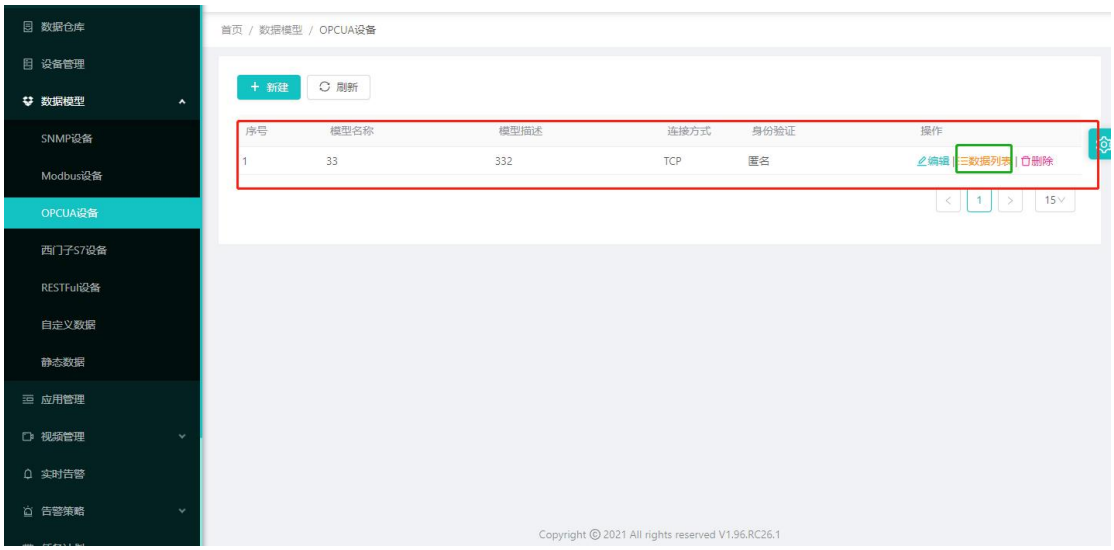
8.7.1.1 步骤 1

登录 Hw-Link，依次点击数据模型->OPC UA 设备->新建

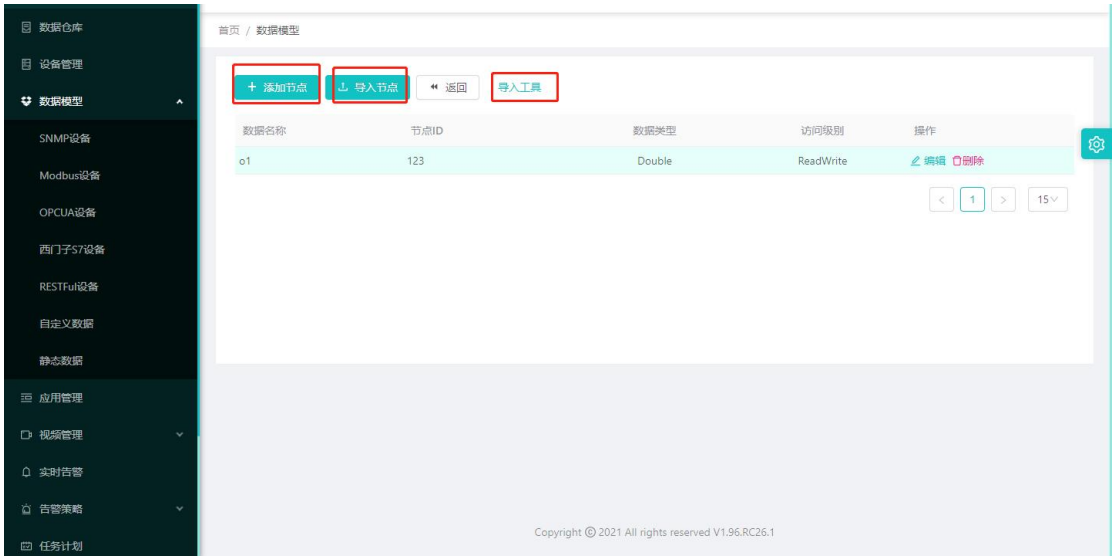
根据设备的实际情况，填写通信参数。这里就讲一个采集数量，这里的意思是 Hw-Link 一次采集的 OPCUA 数据的数量，有些设备可能支持不了大规模的数据采集，这里我们可以填写我们一次采集的数量，来减少设备的数据压力，比如设备共有数据 100 条，Hw-Link 会按照采集数量采集，分 100/30 次采集完成。

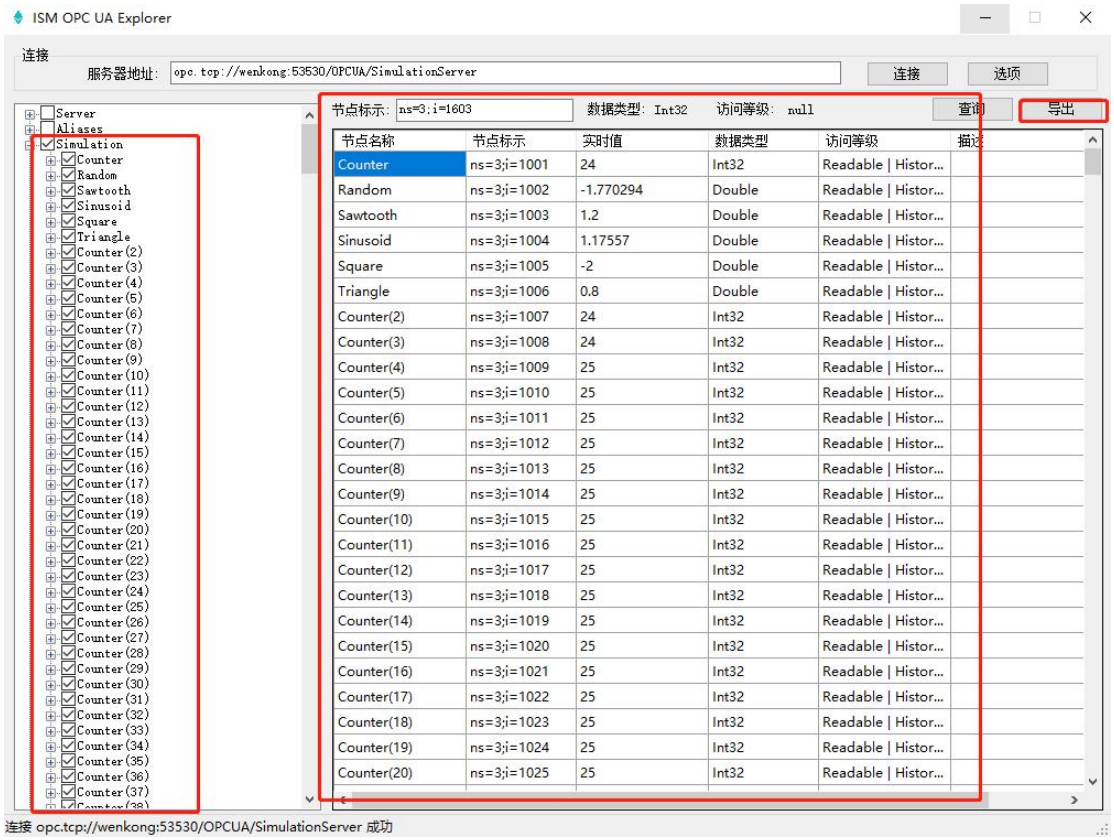
8.7.1.2 步骤 2

找到我们刚才创建的模型。点击数据列表



点击完数据列表后，会跳转到添加 OPC UA 节点的界面，在这里你可以手动添加节点，也可以导入节点，导入节点要配合导入工具使用，导入工具是针对有很多节点需要录入的情况下，可以通过此工具一次导入完成。

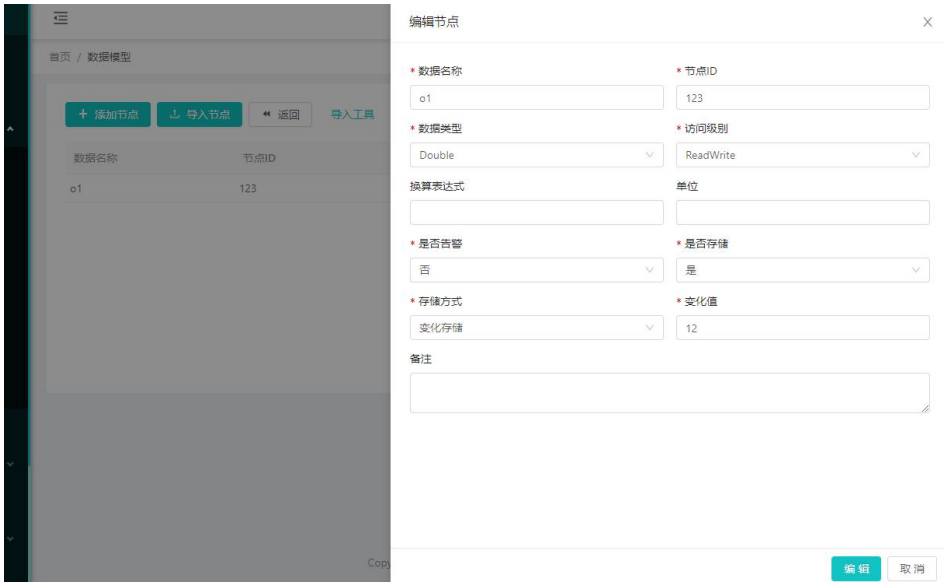




此工具源码开放，需要的联系。

8.7.1.3 步骤 3

数据的修改，可以根据需要，修改添加的节点参数。



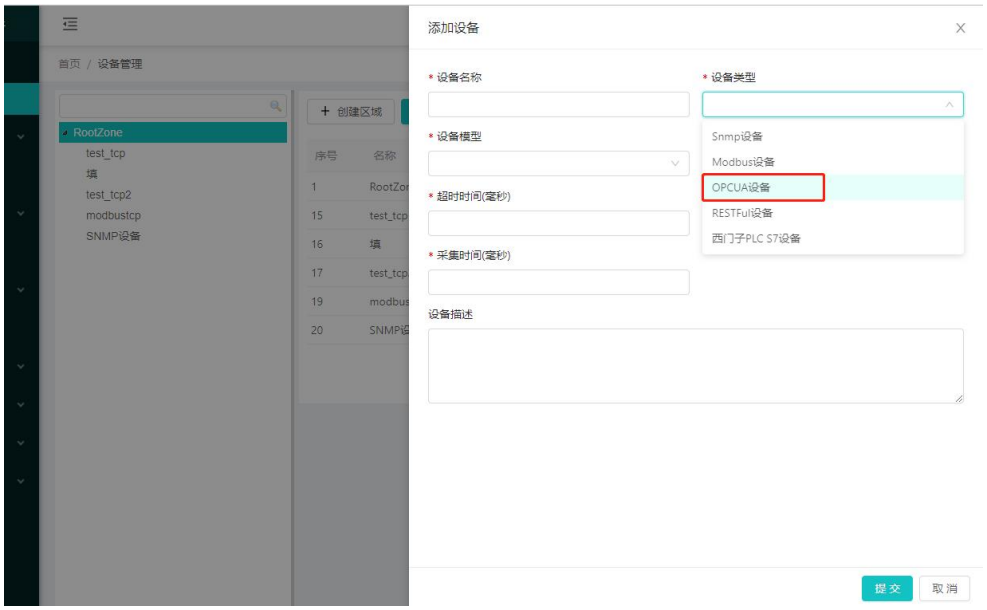
8.7.2 OPC UA 设备的创建

相关信息

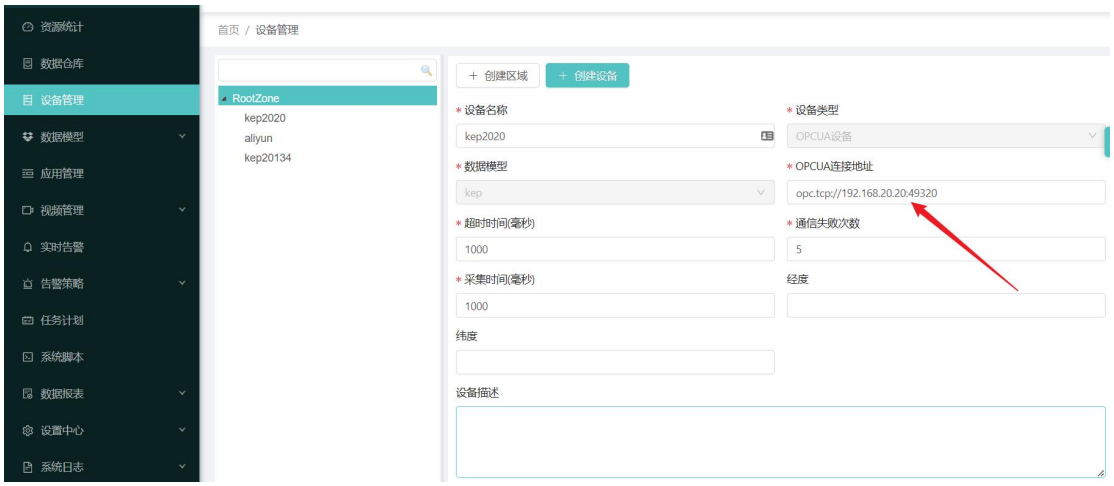
本节讲述 OPCUA 设备的创建

8.7.2.1 步骤 1

点击设备管理，点击创建设备，弹出的页面中设备类型选择 OPCUA 设备，设备模型选择我们刚才创建好的数据模型。

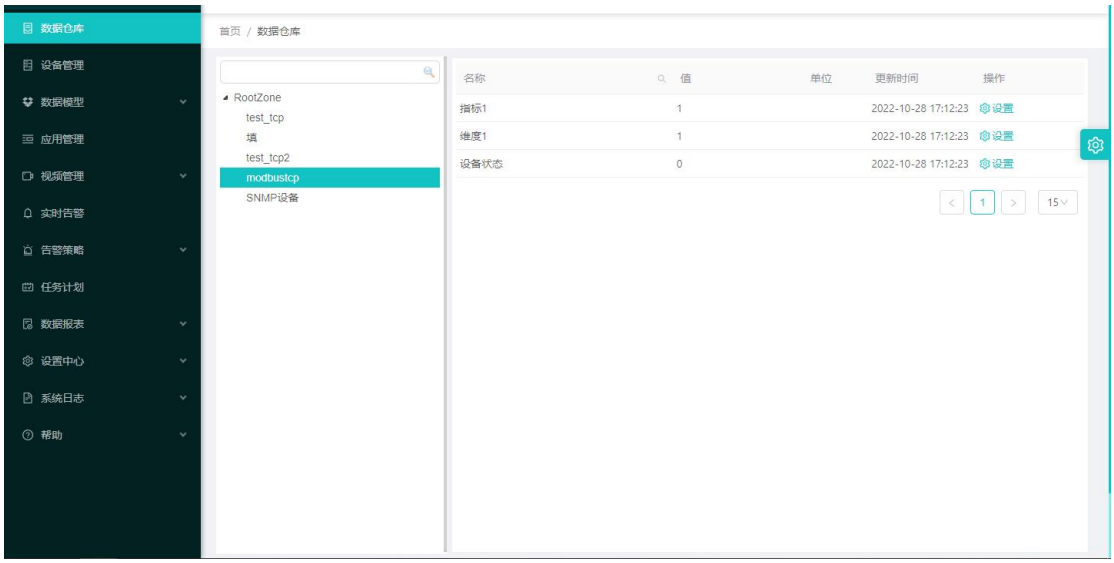


然后根据界面的提示完成设备的创建。以 KepServerEX 软件的 OPC UA Server 为例，填写 OPC UA 连接地址格式为：opc.tcp://192.168.20.20:49320



8.7.2.2 步骤 2

设备创建完成后，点击数据仓库，即可查看设备采集的数据



8.8 RESTFul 协议采集

RESTFUL 是一种网络应用程序的设计风格和开发方式，基于 HTTP，可以使用 XML 格式定义或 JSON 格式定义。RESTFUL 适用于移动互联网厂商作为业务接口的场景，实现第三方 OTT 调用移动网络资源的功能，动作类型为新增、变更、删除所调用资源。

8.8.1 创建 RESTFul 设备模型

相关信息

本节讲述 RESTFul 数据模型的创建。用于 http 客户端往 Hw-Link 推送数据。

RESTFul 数据格式，接口用 JSON 数据格式，下面为格式例子。


```
{

"AccessToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NjI2OTQxODMsInN1YiI6InRva2VuIn
0.pDs3eOvNqfmGPt3pvGzQvmWoXg5lbU7ZnTwVIdCWmls",
  "DeviceFlag":"cf70a4d1-83a4-9267-98cc-193a267655d9",
  "UpdateList":[
    {
      "DataModelFlag":"d651d71b-69af-f675-84be-16a5079a57aa",
      "value":"1"
    },
    {
      "DataModelFlag":"81dcdb3b-5a24-5981-e8ac-49f005d22c7e",
      "value":"11.3"
    }
  ]
}
```


AccessToken 接口的令牌。需要手动创建。

DeviceFlag 设备标识，标识更新的是台设备

UpdateList 数据更新列表

DataModelFlag 为 RESTFul 数据模型里面的数据标识

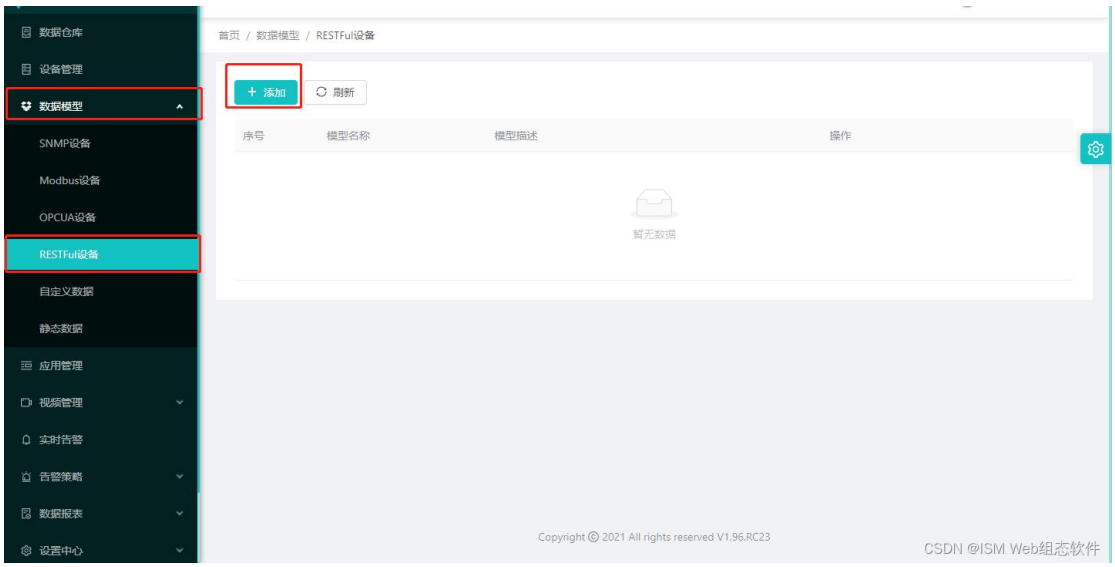
value 准备更新的值，此值必须为字符串类型

请求接口地址 /api/v1/PushDeviceData

8.8.2 模型创建

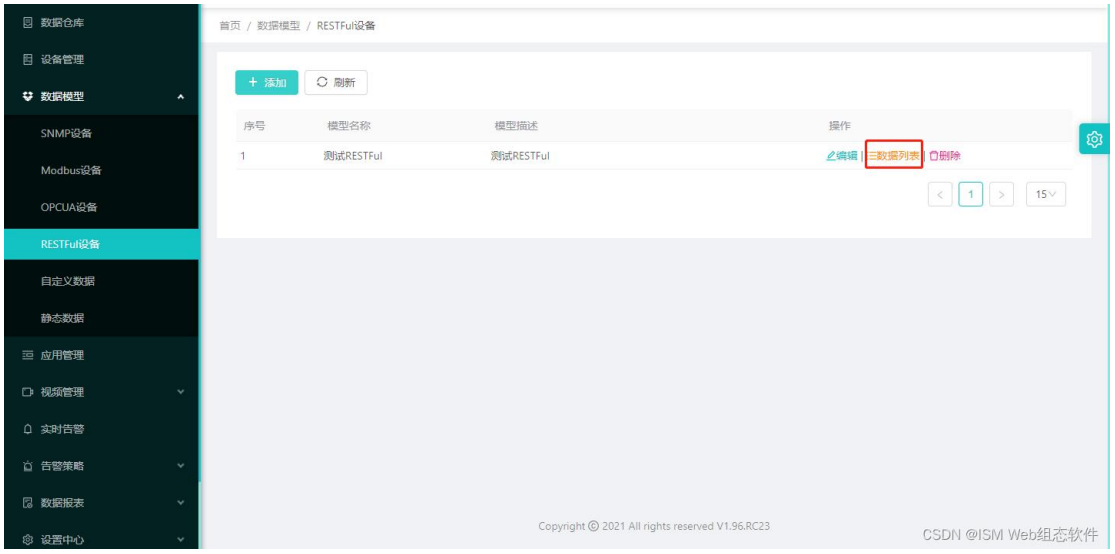
8.8.2.1.建立 RESTFul 设备模型

登录系统后找到数据模型->RESTFul 设备，点击添加按钮，按照界面提示添加 RESTFul 设备数据模型

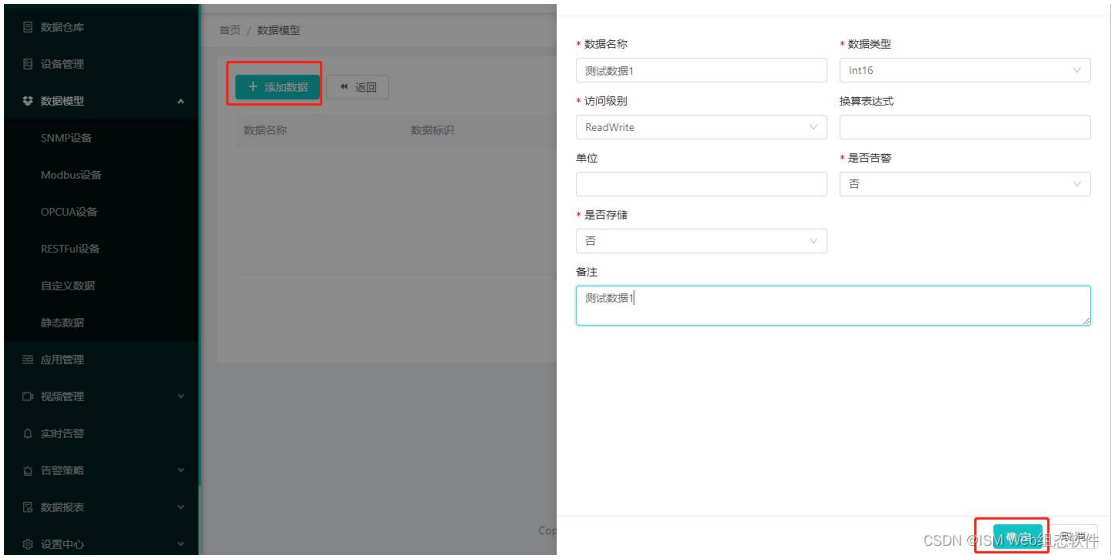


8.8.2.2.添加 RESTFul 数据

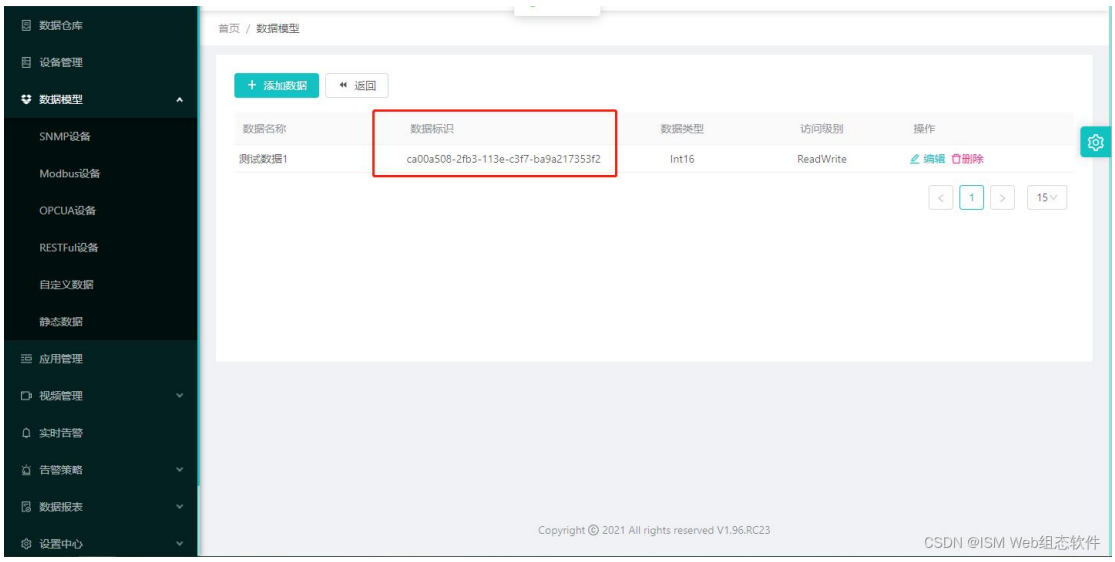
找到刚才添加的数据模型，点击数据列表，进入模型数据添加



添加添加数据，弹窗数据参数界面，根据界面上的提示，添加所需要的数据。

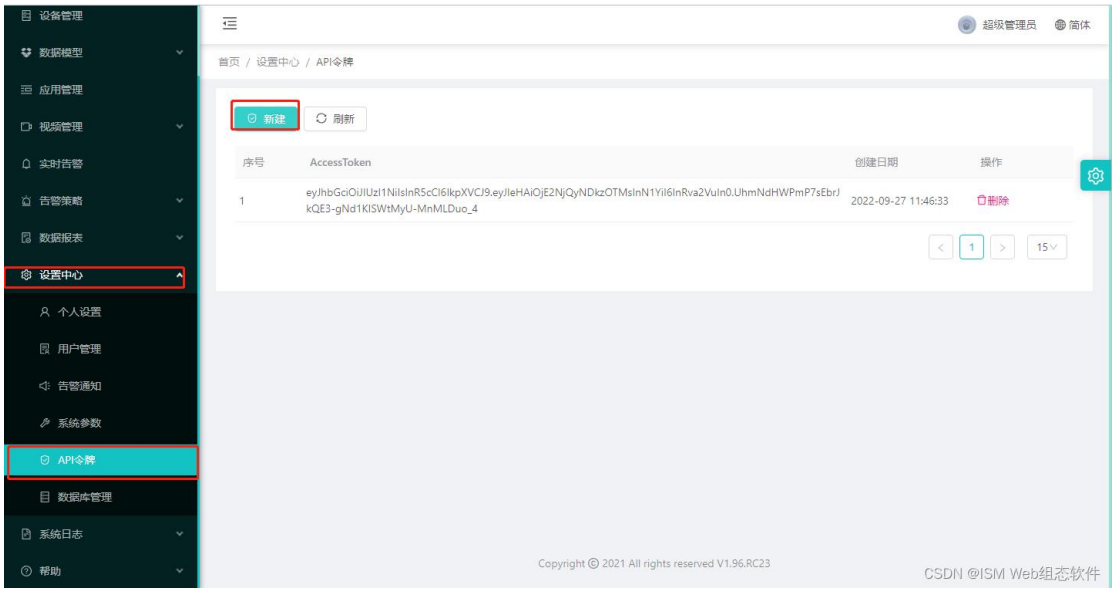


添加完成后列表页会出现数据标识，这个标识后面会用到。



8.8.3 申请接口的令牌

找到设置中心->API 令牌，单击新建按钮，系统会自动分配一个令牌。



8.9 MQTT 协议采集

8.9.1 内置 MQTT 服务配置

8.9.1.1 配置说明

Hw-Link 已经内置了 MQTT 的 broken,配置文件在安装目录下面的 conf 目录里面的 mqtt_broken_config.json 文件

- listeners 下面的 address 是监听的端口，可以根据需要修改
- ledger 下面的 auth 数组存放是连接的用户名和密码，allow true 允许连接，false 禁止连接，可以添加多组用户名和密码
- 其他字段保持默认就可以了
- 配置完成后，重启 Hw-Link 即可

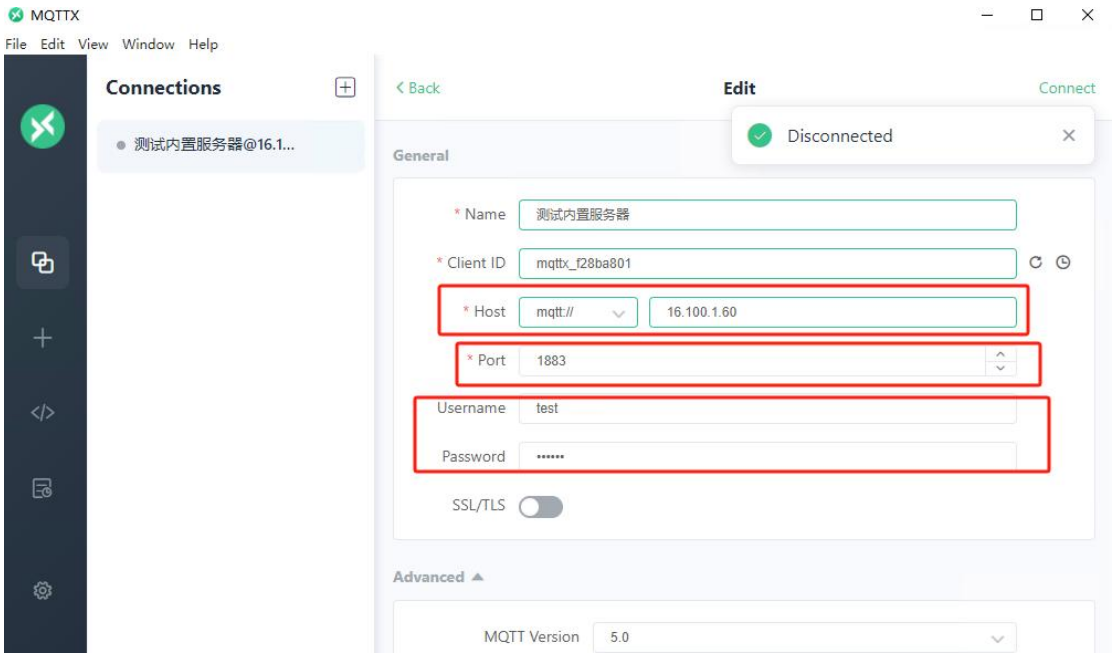
```
{
  "listeners": [
    {
      "type": "tcp",
      "id": "file-tcp1",
      "address": ":1883"
    }
  ],
  "hooks": {
    "auth": {
      "allow_all": false,
      "ledger": {
```

```
        "auth": [
          {
            "username": "test",
            "password": "123456",
            "allow": true
          }
        ],
        "acl": [
          {
            "remote": "127.0.0.1:*"
          },
          {
            "username": "melon",
            "filters": null,
            "melon/#": 3,
            "updates/#": 2
          }
        ]
      }
    }
  }
}
```

8.9.1.2 使用 MQTTX 连接测试

打开下载地址:安装包下载地址

然后根据界面的提示完成 MQTTX 的安装，安装完成后根据提示新建连接，配置如下图,如果连接失败，请检查系统是否有防火墙，允许 1883 端口通过



8.9.2 配置 Hw-Link 的 MQTT 连接参数

8.9.2.1 配置说明

- Hw-Link 的 MQTT 配置文件路径在安装目录下面 conf 目录，配置文件为 mqtt.conf，Hw-Link 实时检测配置的更新状态。
- isEnable 改为 true,开启 MQTT 连接
- 修改 BrokerHost 的连接地址,如果使用内置的 broken，配置为"127.0.0.1",连接第三方服务，根据场景，修改连接地址。
- BrokerPort，连接的端口，默认 1883
- UserName 连接的用户名，如果没有留空
- PassWord 连接密码，没有就留空
- SubscribeTopic 订阅主题，必须要包含${ClientID}参数, 设备要按照此主题推送数据
- PublishTopic 发布主题，必须要包含${ClientID}参数，Hw-Link 下发的数据，通过此主题推送给设备

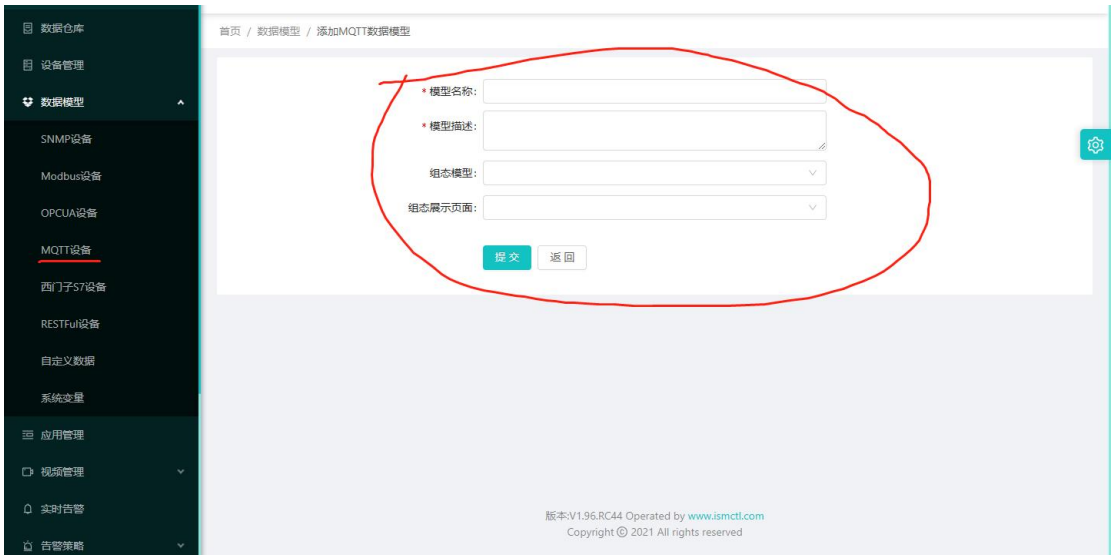
8.9.2.2 样例配置

```
mqttCloudPlat=1
isEnable=true        [MQTT]
BrokerHost="127.0.0.1"
BrokerPort=1883
UserName="Hw-Link"
PassWord="123456"
ClientID="Hw-Link"
TLS=false
certPath=conf/x509/root.pem
SubscribeTopic=/sys/${ClientID}/post
PublishTopic=/sys/${ClientID}/pub
```

8.9.3 创建 MQTT 的数据模型

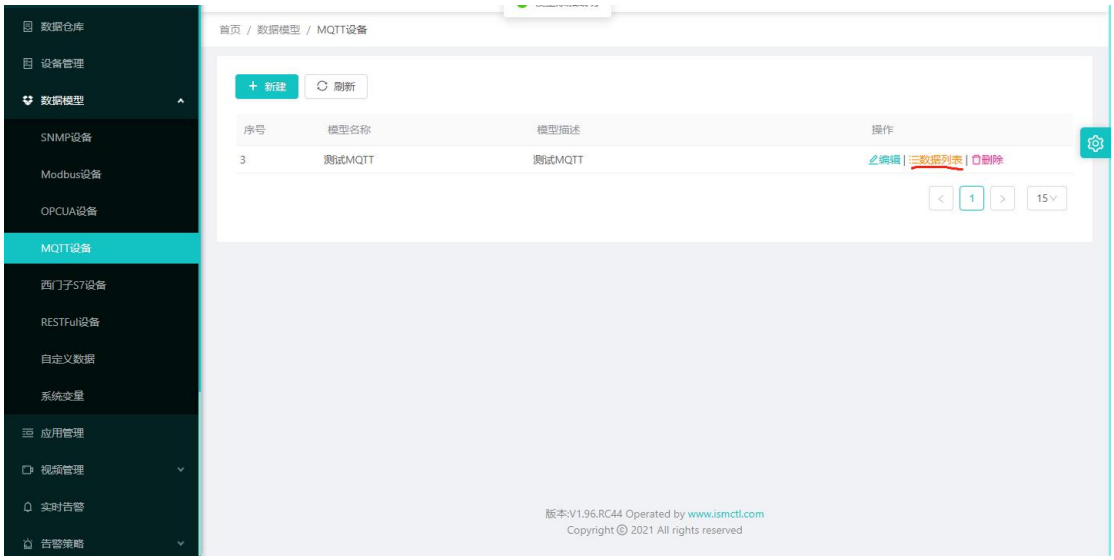
8.9.3.1 步骤 1

登录 Hw-Link，依次点击数据模型->MQTT 设备->新建
根据页面提示填写参数

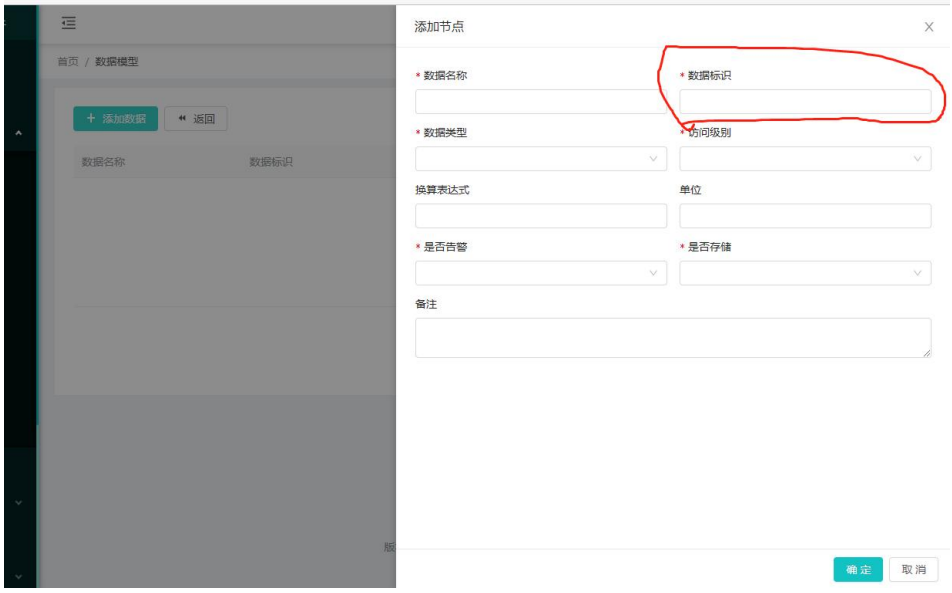


8.9.3.2 步骤 2

找到我们刚才创建的模型。点击数据列表

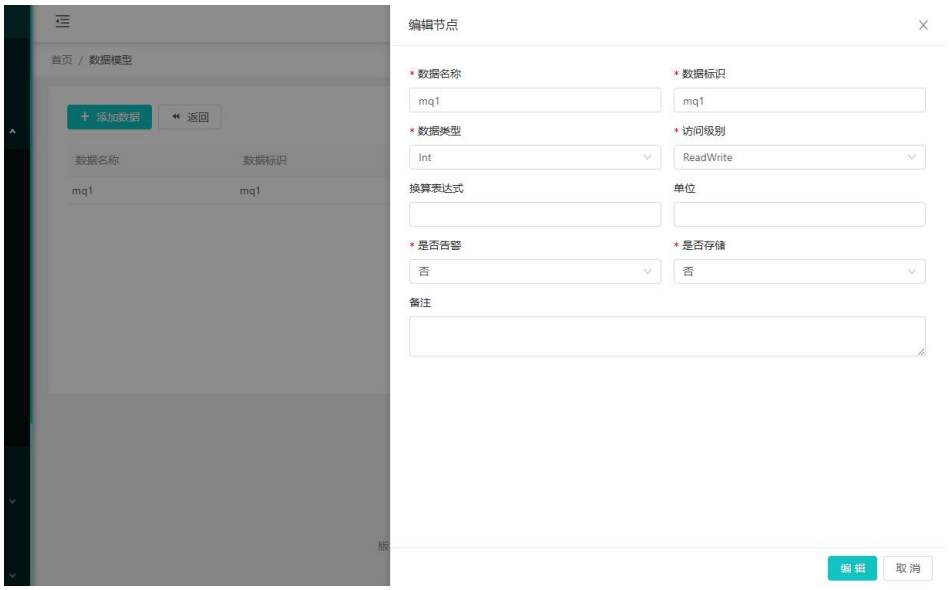


点击完数据列表后，会跳转到添加 MQTT 数据的界面，点击添加数据按钮，弹出的添加数据框中数据标识要跟通信格式中的名称一致



8.9.3.3 步骤 3

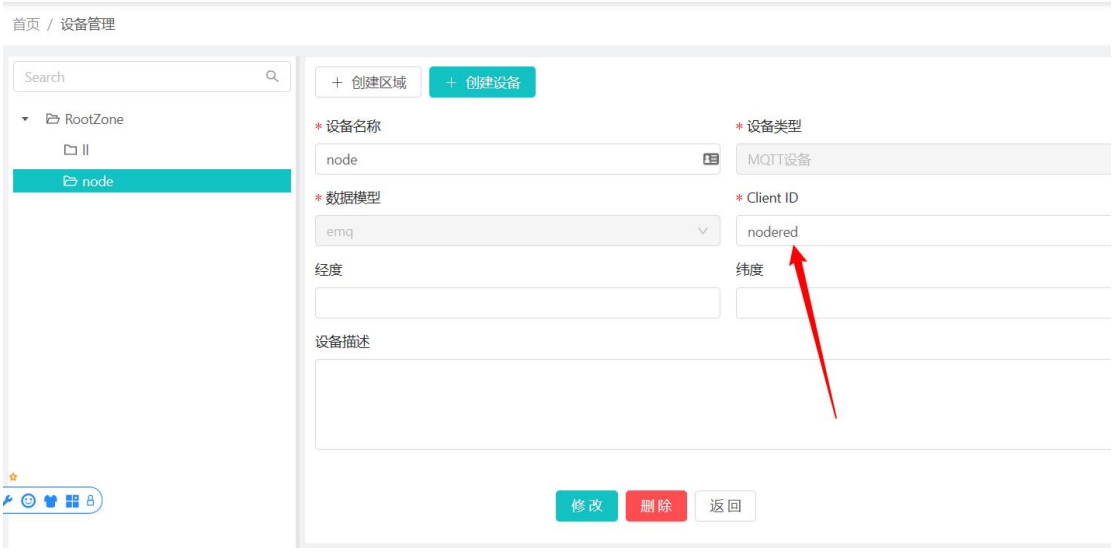
数据的修改，可以根据需要，修改添加的节点参数。



8.9.4 创建 MQTT 设备

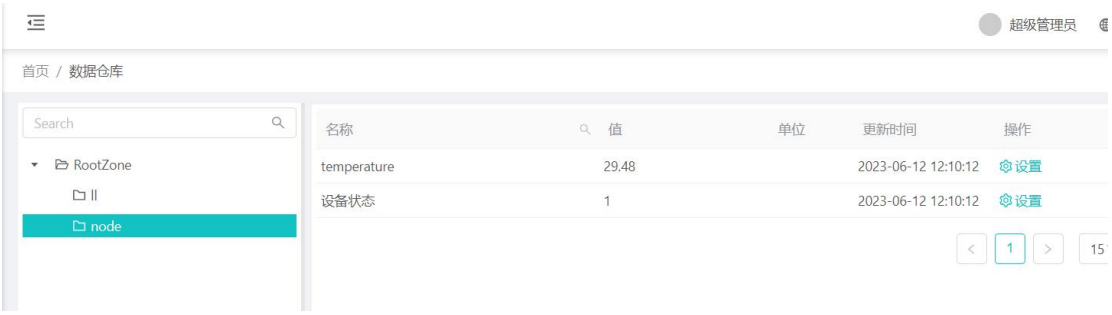
8.9.4.1 步骤 1

- 点击设备管理，点击创建设备，弹出的页面中设备类型选择 MQTT 设备，设备模型选择我们刚才创建好的数据模型。
- 然后根据界面的提示完成设备的创建，此处的 Client ID 要求跟设备连接的 ClientID 要一致,不然会导致设备的在线和离线状态出错



8.9.4.2 步骤 2

- 设备创建完成后，点击数据仓库，即可查看设备采集的数据



8.9.5 数据标识详细说明

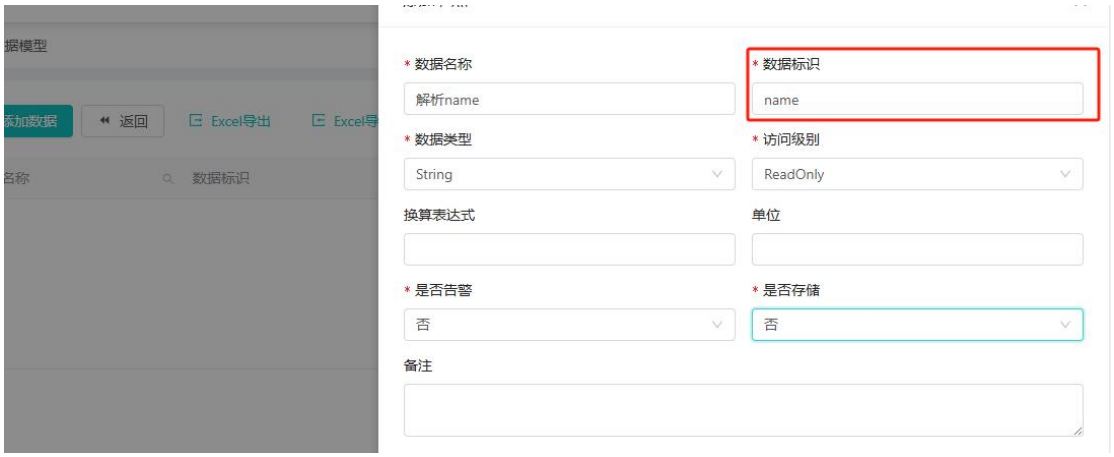
8.9.5.1 数据格式说明

- 传输的协议格式是 JSON 格式，JSON 只有 UTF-8 文件格式，如果遇到中文，一定要转 UTF-8 格式。
- 持复杂的 json 格式解析，通过路径查找。
- 支持数组索引查找

8.9.5.2 简单格式解析

{"id": 2,"name": "Bob"}

这种简单的 json 对象格式，数据模型的数据标识直接写 name，即可解析推送的数据中的 name 的数据值



8.9.5.3 复杂格式嵌套对象格式解析

```
{
    "id": 2,
    "name": "Bob",
    "address":
        {
            "city": "Los Angeles",
            "zip": "90001"
        },
    "obj1":{
        "obj2":{
            "obj3":{
                "name":"测试多层嵌套"
            }
        }
    }
```

}

如果要解析的数据，嵌套很多层,这时我们就需要使用路径查找，比如我们现在需要解析 obj3 这个对象里面的 name 的数据值，那么我们就要填写完整的嵌套路径 ，数据标识应该这样填写 obj1.obj2.obj3.name 以.分割路径。

8.9.5.4 数组格式解析

```
{
    "users": [
        {"id": 1, "name": "Alice", "age": 30, "city": "New York"},
        {"id": 2, "name": "Bob", "age": 25, "city": "Los Angeles"},
        {"id": 3, "name": "Charlie", "age": 35, "city": "Chicago"},
        {"id": 4, "name": "David", "age": 28, "city": "New York"},
        {"id": 5, "name": "Eve", "age": 22, "city": "Los Angeles"}
    ],
    "temperatures":[19, 90.9, 7, 67.5]}
```

这个数组格式的数据，我们需要明确我们要解析的索引号，这时我们想要获取第一个数据中的 name 的数值，那么数据标识应该配置为 users.[0].name，意思就是获取 users 字段下面第一个数据下面的 name，依次类推，第二

个就是 users.[1].name,如果我们要获取 temperatures 数组中的第一个，我们应该这样填，temperatures.[0]

8.9.6 EMQX 服务配置

8.9.6.1 下载并安装 EMQX 服务器

相关信息

Hw-Link 支持 emqx 的 MQTT broker。如果使用 Mosquitto 等，可以正常通讯，但是无法获取设备上线信息。推荐使用 Hw-Link 自带的 MQTT broker

8.9.6.1.1 下载并安装 EMQX 服务器

步骤 1

访问 Eqmx 官网 https://www.emqx.io/zh

步骤 2

按照官网的介绍，选择自己合适的版本下载安装，具体请参见官网文档。

8.9.6.1.2 配置 EMQX 服务器

步骤 1.访问服务器面板

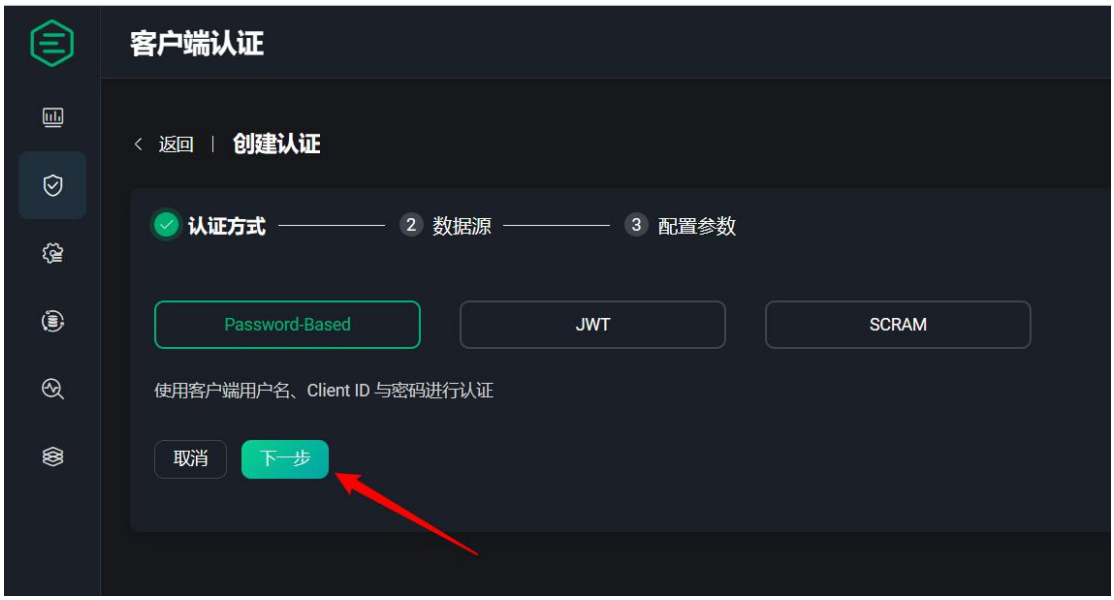
使用浏览器访问 18083 端口，默认用户名和密码是 admin public

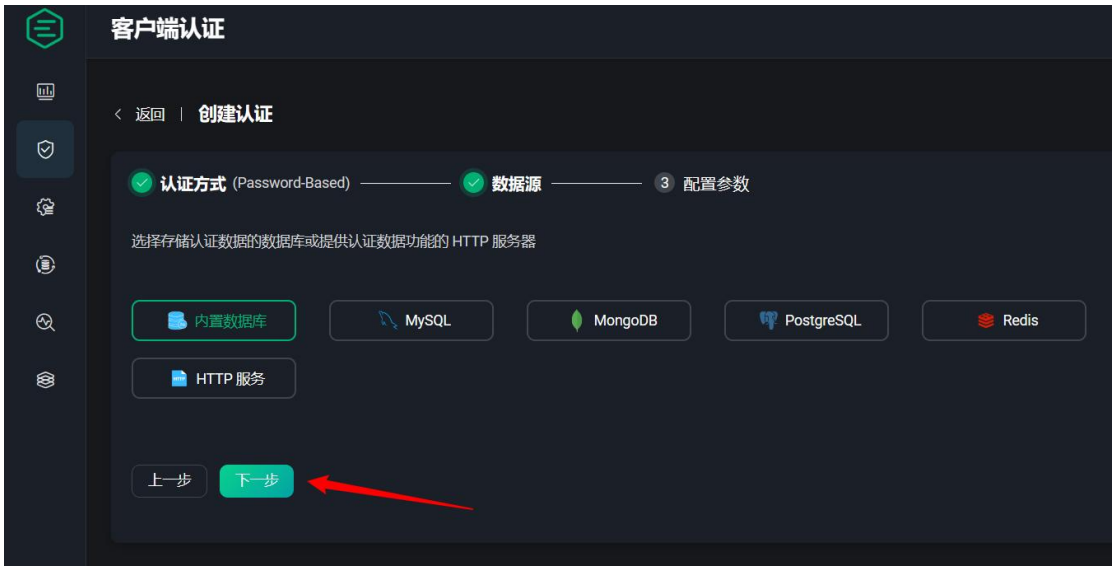
步骤 2.增加 Hw-Link 用户

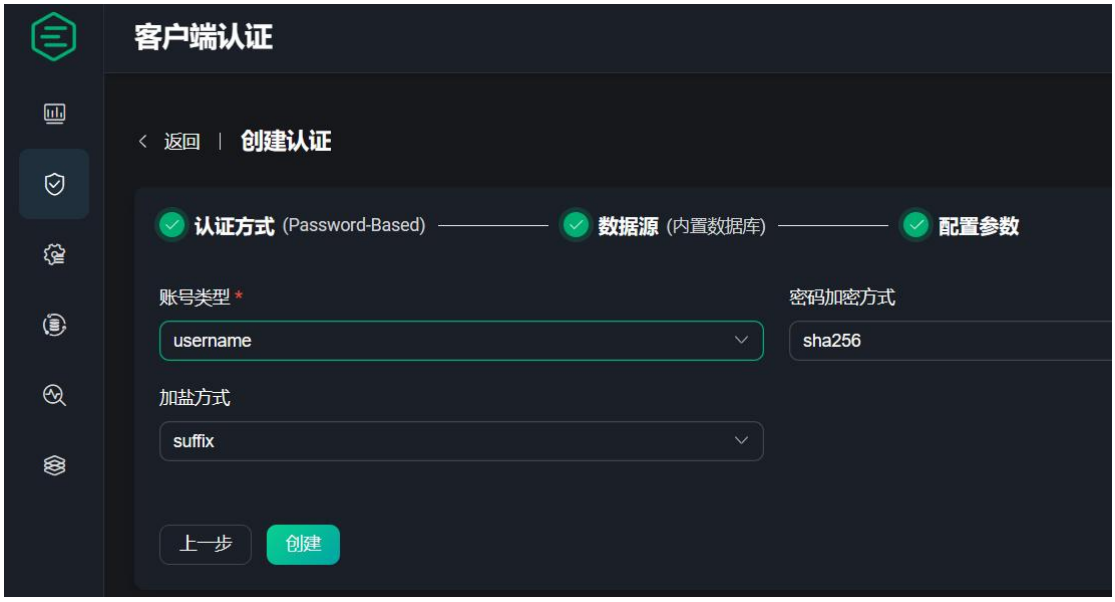
然后进入客户端认证


客户端认证
客户端认证
客户端授权
黑名单

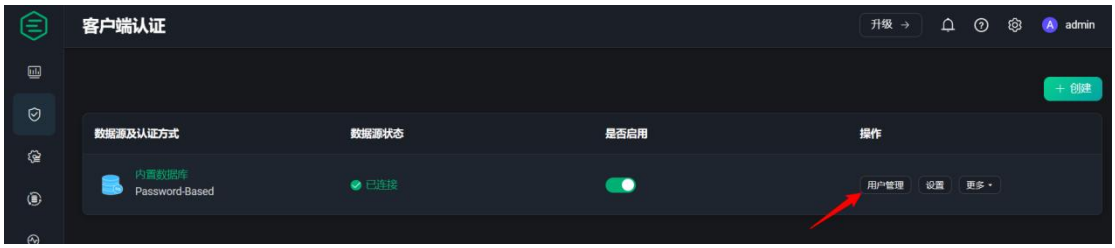

客户端认证
升级
admin
创建
数据源及认证方式
数据源状态
是否启用
操作
暂无数据

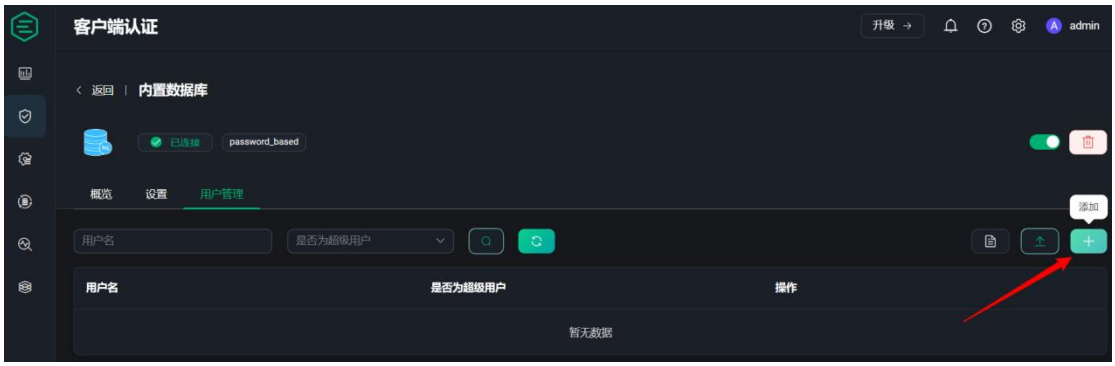

客户端认证
返回 | 创建认证
认证方式
2 数据源
3 配置参数
Password-Based
JWT
SCRAM
使用客户端用户名、Client ID 与密码进行认证
取消
下一步

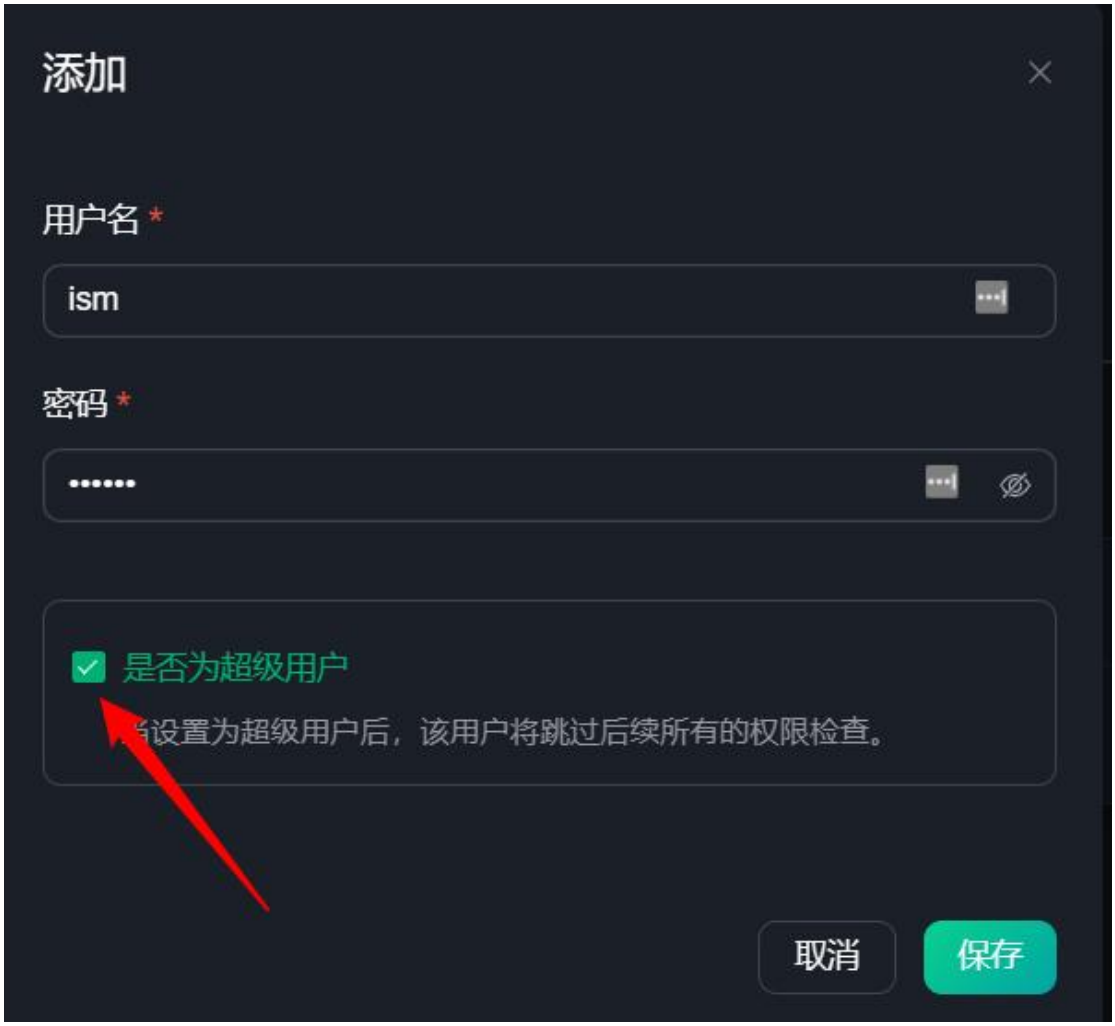

客户端认证
返回 | 创建认证
认证方式 (Password-Based)
数据源
3 配置参数
选择存储认证数据的数据库或提供认证数据功能的 HTTP 服务器
内置数据库
MySQL
MongoDB
PostgreSQL
Redis
HTTP 服务
上一步
下一步


客户端认证
返回 | 创建认证
认证方式 (Password-Based)
数据源 (内置数据库)
配置参数
账号类型 *
username
密码加密方式
sha256
加盐方式
suffix
上一步
创建

新建 SIM 用户.注意，一定要勾选为超级用户，免得后续的权限设置。







8.9.6.2 在 Emqx 上配置 Hw-Link 的权限

相关信息

Hw-Link 判断设备上下线的消息需要订阅 Emqx 的系统主题，此节是为了展示如何配置 Hw-Link 的权限，用于订阅系统主题。

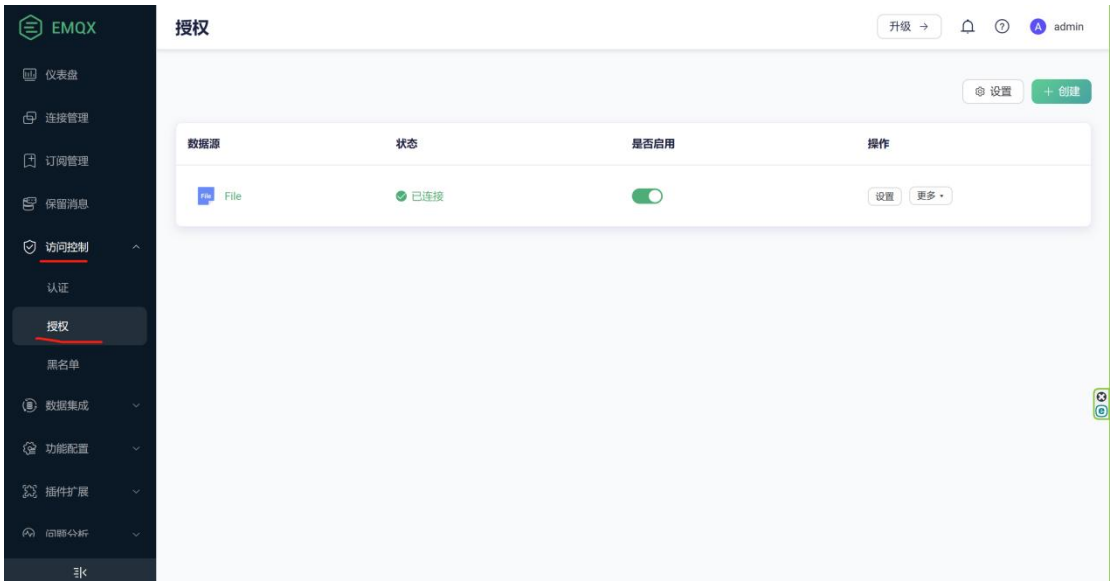
8.9.6.2.1 下载并安装 emqx 服务器

步骤 1

打开 Emqx 的网页，如 127.0.0.1：18083，然后登录

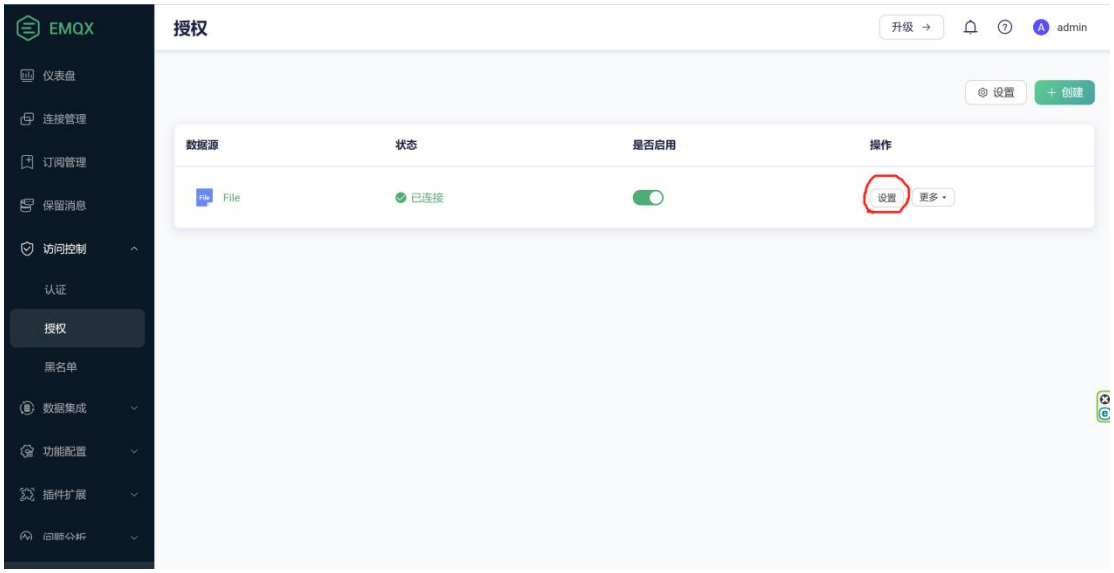
步骤 2

左侧菜单找到访问控制->授权



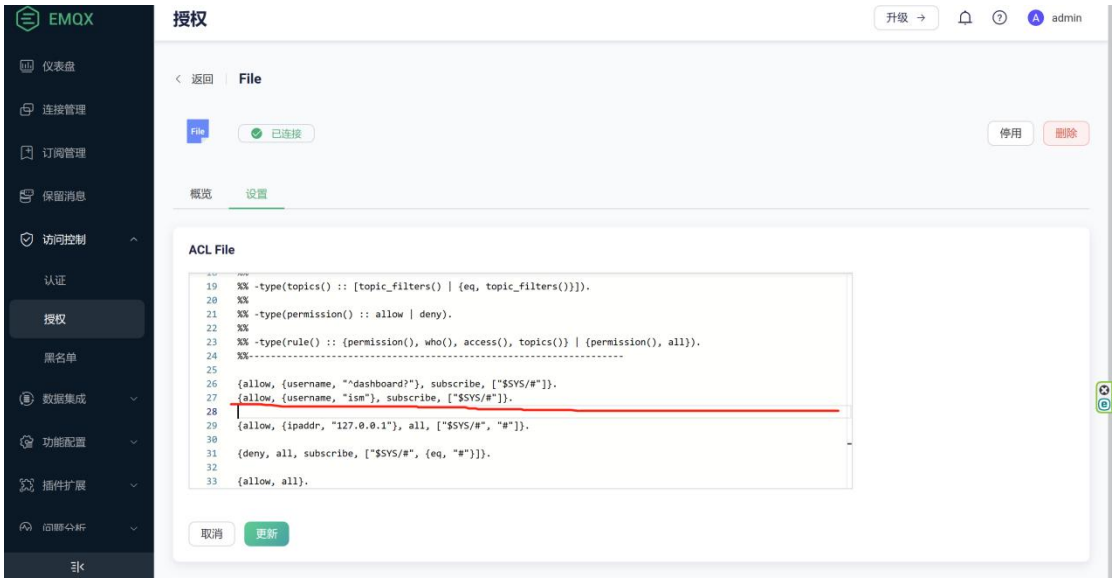
8.9.6.2.2 设置权限

点击设置

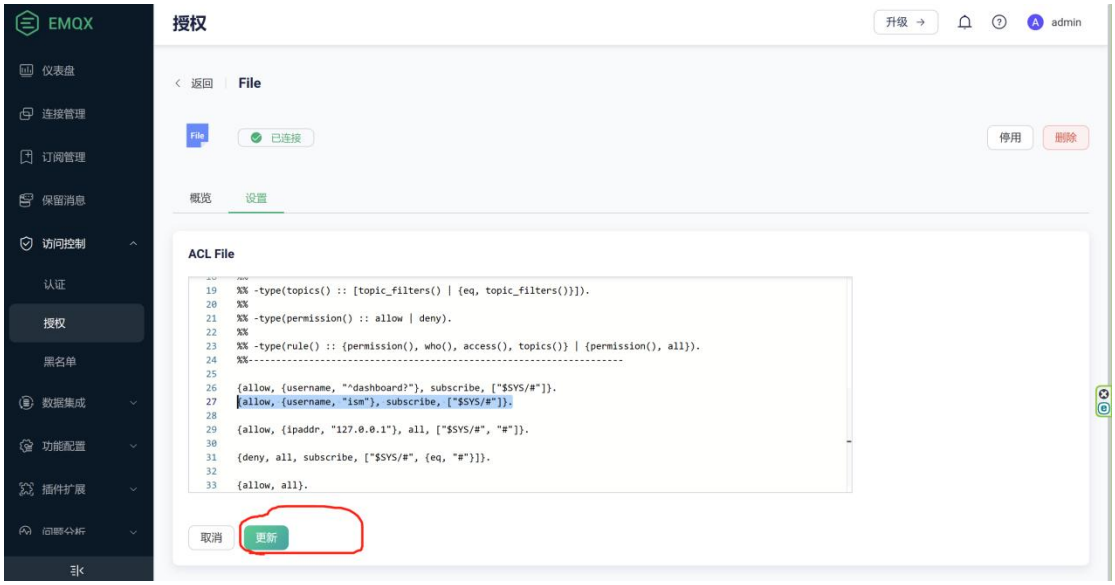


添加 {allow, {username, "Hw-Link"}, subscribe, ["$SYS/#"]}.到文件中

这里的用户名 Hw-Link，是 Hw-Link 配置文件中的用户名，实际根据配置修改



点击更新按钮即可



8.9.6.2.3 完成

上述步骤完成后，重启 Hw-Link，然后重启 Mqtt 的客户端即可

8.9.7 MQTT 常见问题

8.9.7.1 MQTT 设备状态问题

- Hw-Link 根据用户配置设备 ID，进行设备上下线的判断，如果使用内置的服务，检查设备连接时填写的连接 ID，这个 ID 要跟配置的保持一致。
- 使用 emqx 服务的，因为 emqx 启动后，设备连接成功后，这时才启动 Hw-Link，Hw-Link 判断不出来，这时 emqx 已经把这条消息发布了，这时你可以断开设备连接，让设备重新连接就可以了。

8.9.7.2 MQTT 设备数据解析不出来

- 检查 Hw-Link 的 mqtt.conf 配置文件是否启用了 MQTT，默认是没有启用的
- 查看设备发布的主题是否跟 Hw-Link 订阅的主题一致。默认是/sys/${ClientID}/post,如果连接的设备 ID 是 mqttx_80e506b8，那么设备应该向/sys/mqttx_80e506b8/post 推送设备的数据给 Hw-Link
- 如果主题没有问题，要检查配置的数据标识跟推送的数据是否一致，请参考数据标识说明章节
- 数据标识没有问题时，就要检查推送的数据类型跟数据模型里面配置的数据类型是否一致
- 如果数据中带有中文，一定要转为 utf-8 格式

8.10 西门子 S 协议采集

8.10.1 创建西门子 PLC S7 数据模型

相关信息

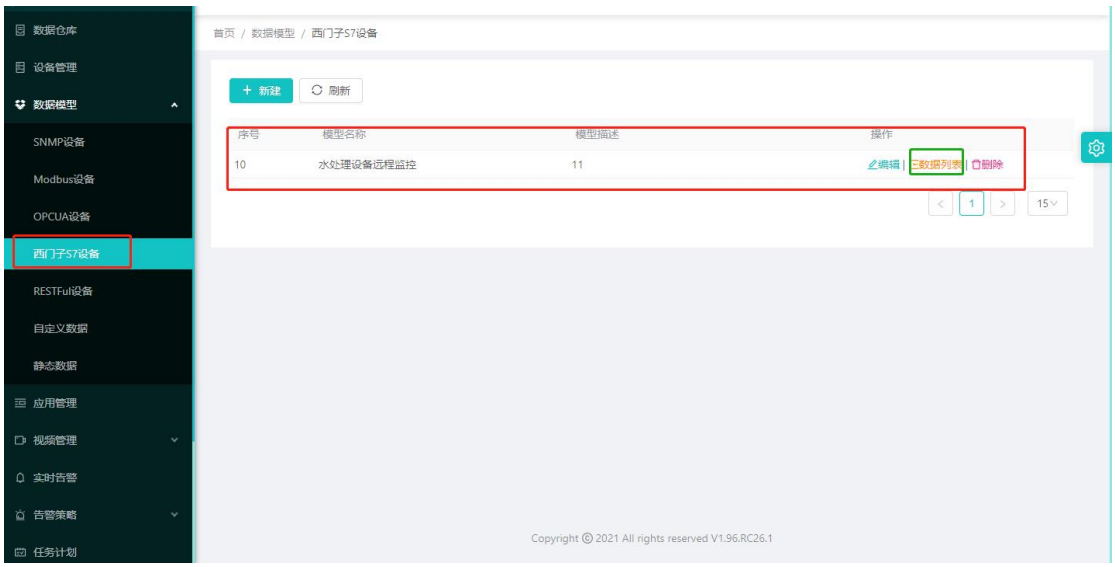
本节讲述西门子 S7 数据模型的创建

8.10.1.1 步骤 1

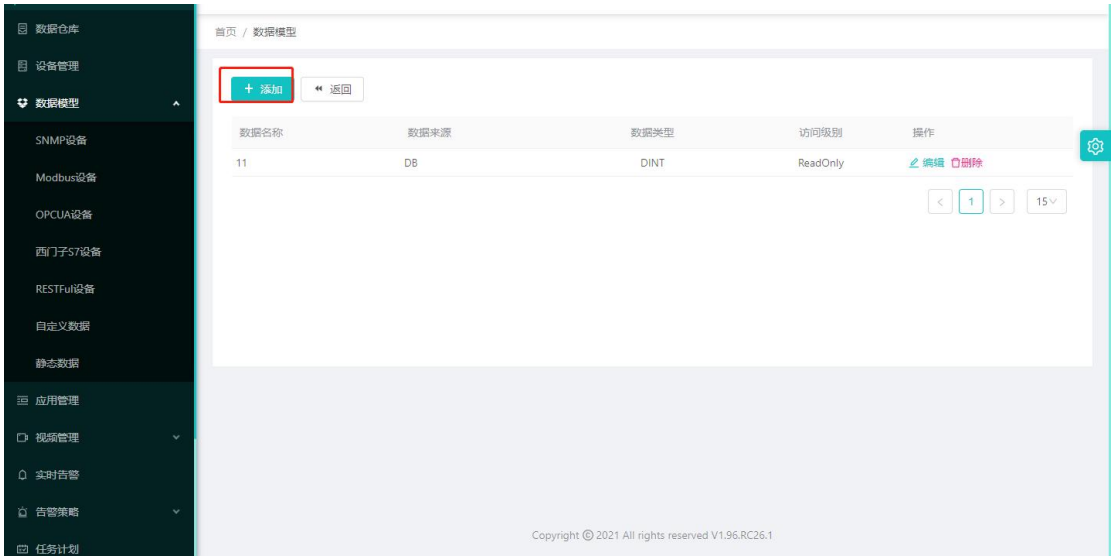
登录 Hw-Link，依次点击数据模型->西门子 S7 设备->新建
按照界面的提示，完成模型的创建

8.10.1.2 步骤 2

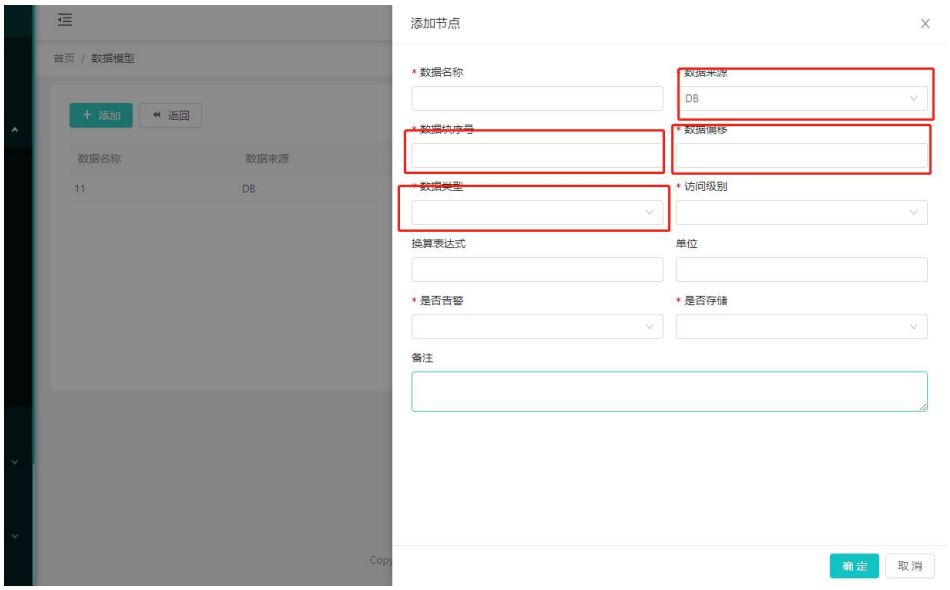
找到我们刚才创建的模型。点击数据列表



点击添加按钮



这里注意一下，数据偏移，如果数据的偏移没有小数点，可以直接填整数，如果有小数点，可以填写小数点，Hw-Link 会根据小数点的偏移读取数据。其他的数据参数参考前言里面的。



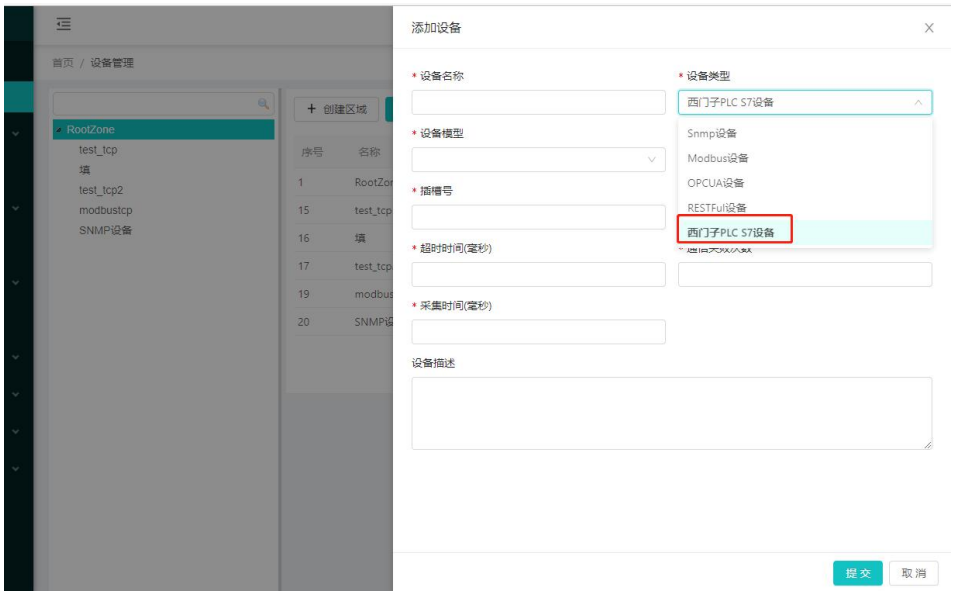
8.10.2 创建西门子 PLC S7 设备

相关信息

本节讲述西门子设备的创建

8.10.2.1 步骤 1

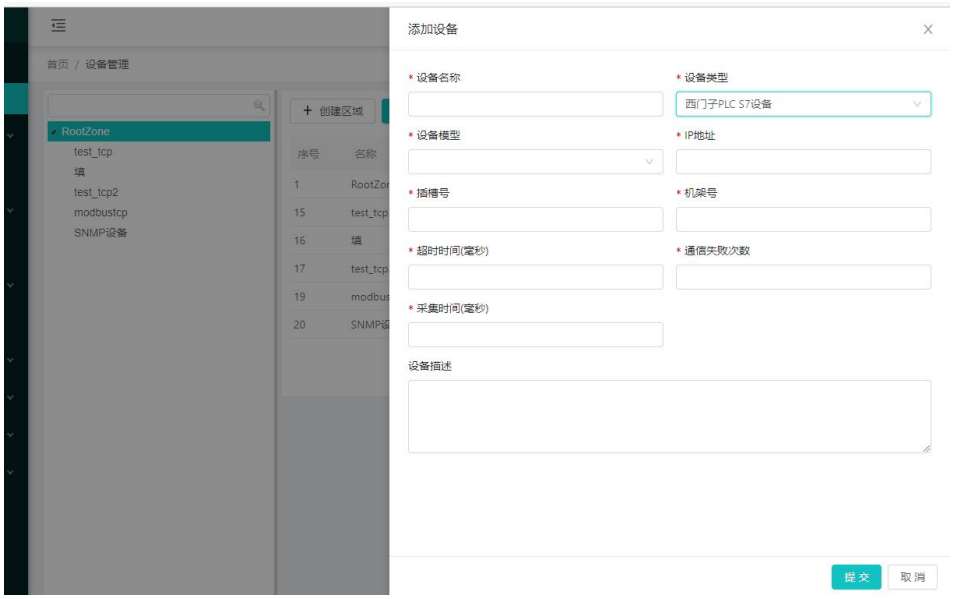
点击设备管理，点击创建设备，弹出的页面中设备类型选择西门子 PLC S7 设备，设备模型选择我们刚才创建好的数据模型。



然后根据界面的提示完成设备的创建,这里注意,西门子的 PLC 默认的通信端口是 102，如果 PLC 不是默认的端口，这里 IP 地址可以带端口，比如 192.168.1.5:203,以:号分割

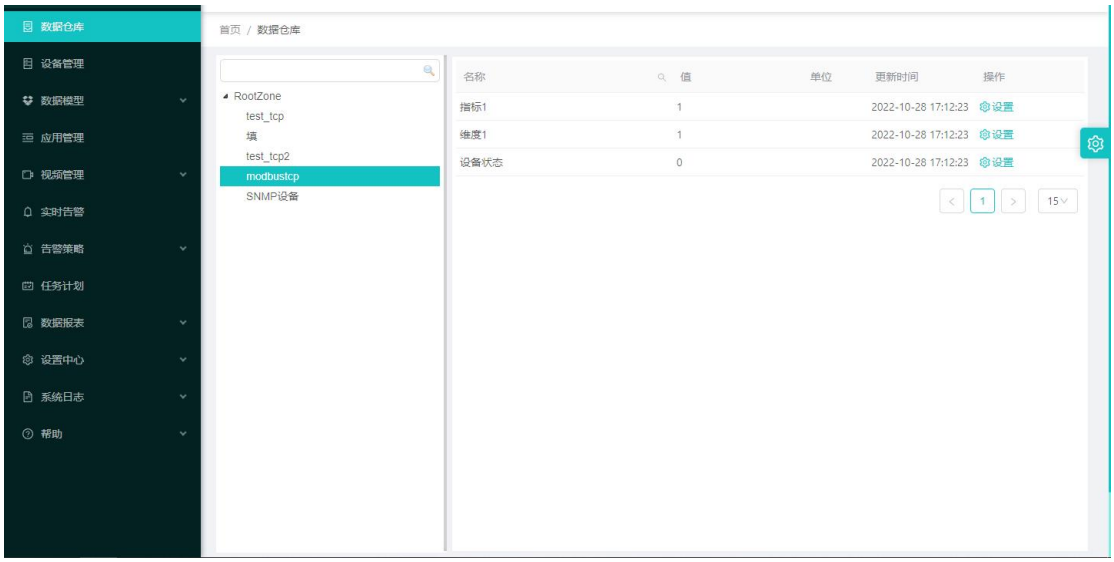
机架号：一般为 0 插槽号：1

S7-200Smart 的机架号是 0，插槽号 1



8.10.2.2 步骤 2

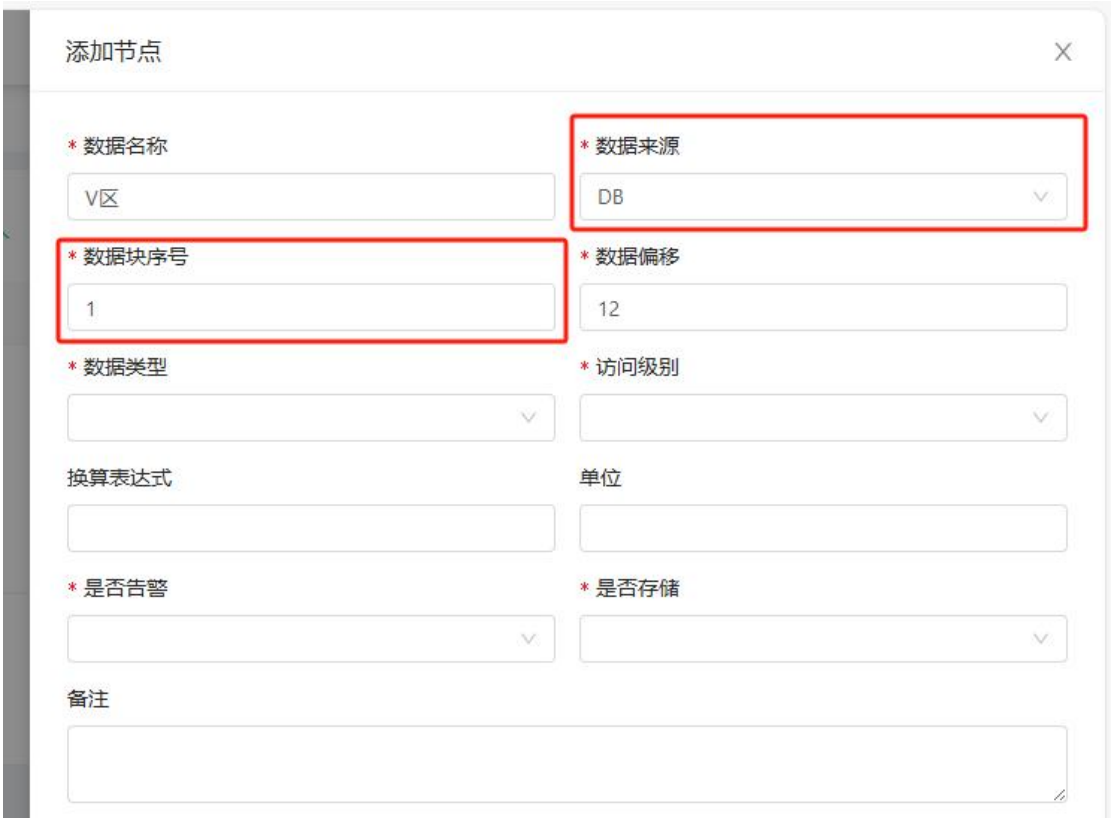
设备创建完成后，点击数据仓库，即可查看设备采集的数据



8.10.3 S7 协议常见问题

是否可以连接 smart200

- 可以
- 200 的 V 区就是 DB1,模型配置里面选择 DB 区



8.11 Hw-Link 组态软件的自定义数据

相关信息

本节讲述自定义数据的创建

8.11.1 简介

自定义数据是数据模型的额外补充，比如有的数据模型里面的数据需要复杂的额外运算才能表达出正在的含义，这时就需要使用自定义数据对此数据进行补充。

8.11.2 自定义数据支持哪些运算

8.11.2.1 算数、比较和逻辑运算

+，-，/，*，&，|，^，**，%，>>，<< 加减乘除，按位与，按位或，异或，乘方，取模，左移和右移；
>>，=，<<，===，!==，~ !，~，=~为正则匹配，!~为正则不匹配；
|| 逻辑或 && 逻辑与。

8.11.2.2 常量

数字常量，将数字都作为 64 位浮点数处理；
字符串常量，字符串用单引号'；
布尔常量：true、false。

8.11.2.3 其他

圆括号可以改变计算优先级；
三目运算符：? :

8.11.3 使用说明

8.11.3.1 取位运算

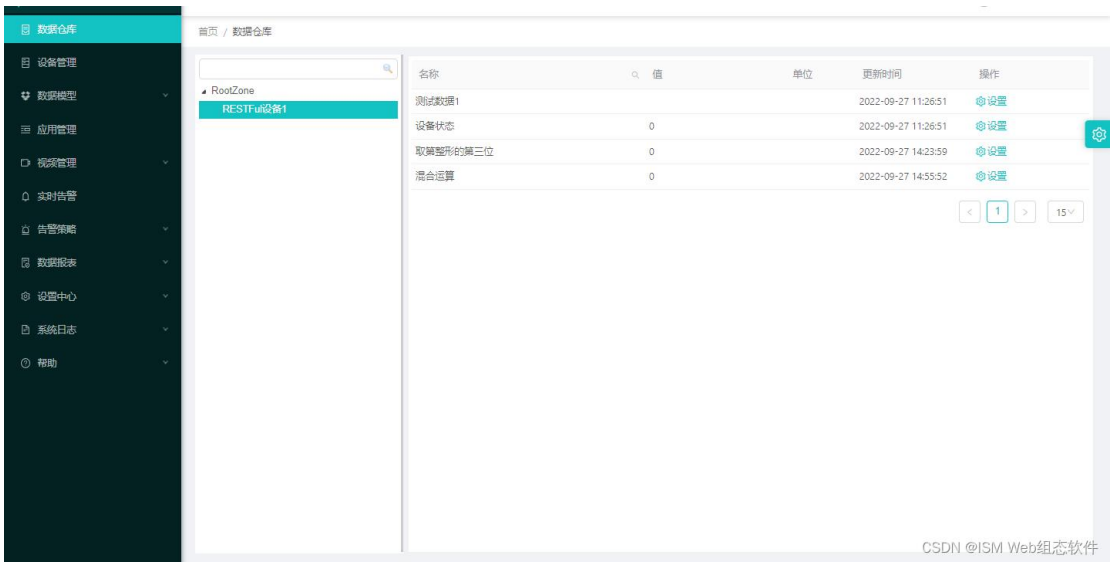
换算表达式:({val}>>3)&1
{val}是数据模型里的数据变量，表示数据模型里的数据值右移 3 位，然后在与 1，就可以取到数据的第三位的 bit 值

8.11.3.2 复杂数据运算

换算表达式:{val}*36.6+({val}*3.6+69)
表达式中可以出现多个{val}标识符

8.11.3.3 查看自定义数据结果

新增的自定义数据会自动添加到所有绑定数据来源的设备中，



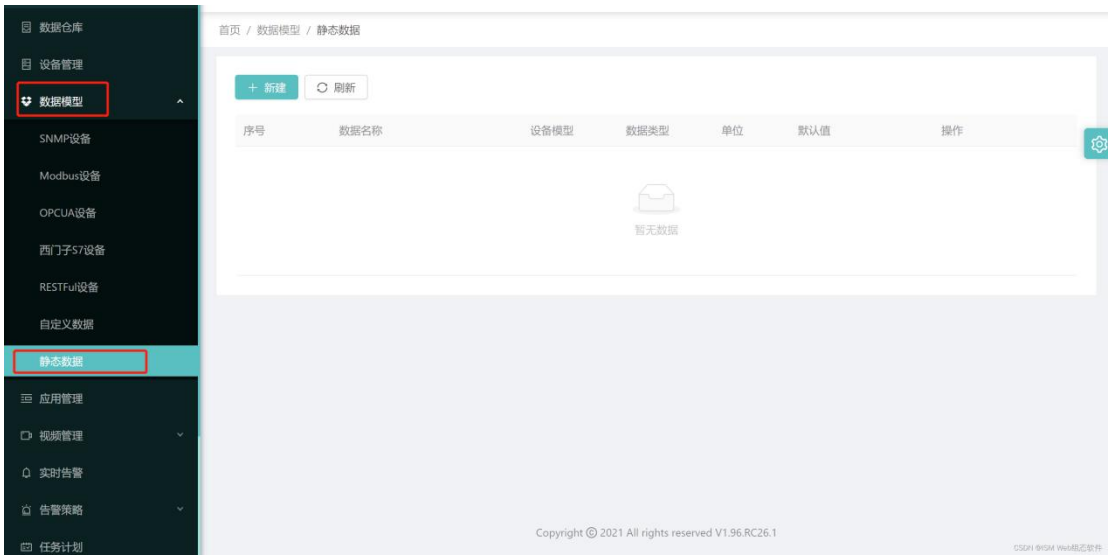
8.12 Hw-Link 组态软件的系统变量

什么是系统变量?

系统变量就是 Hw-Link 提供给用户的数据变量，用户可以设置或者获取此变量的值。

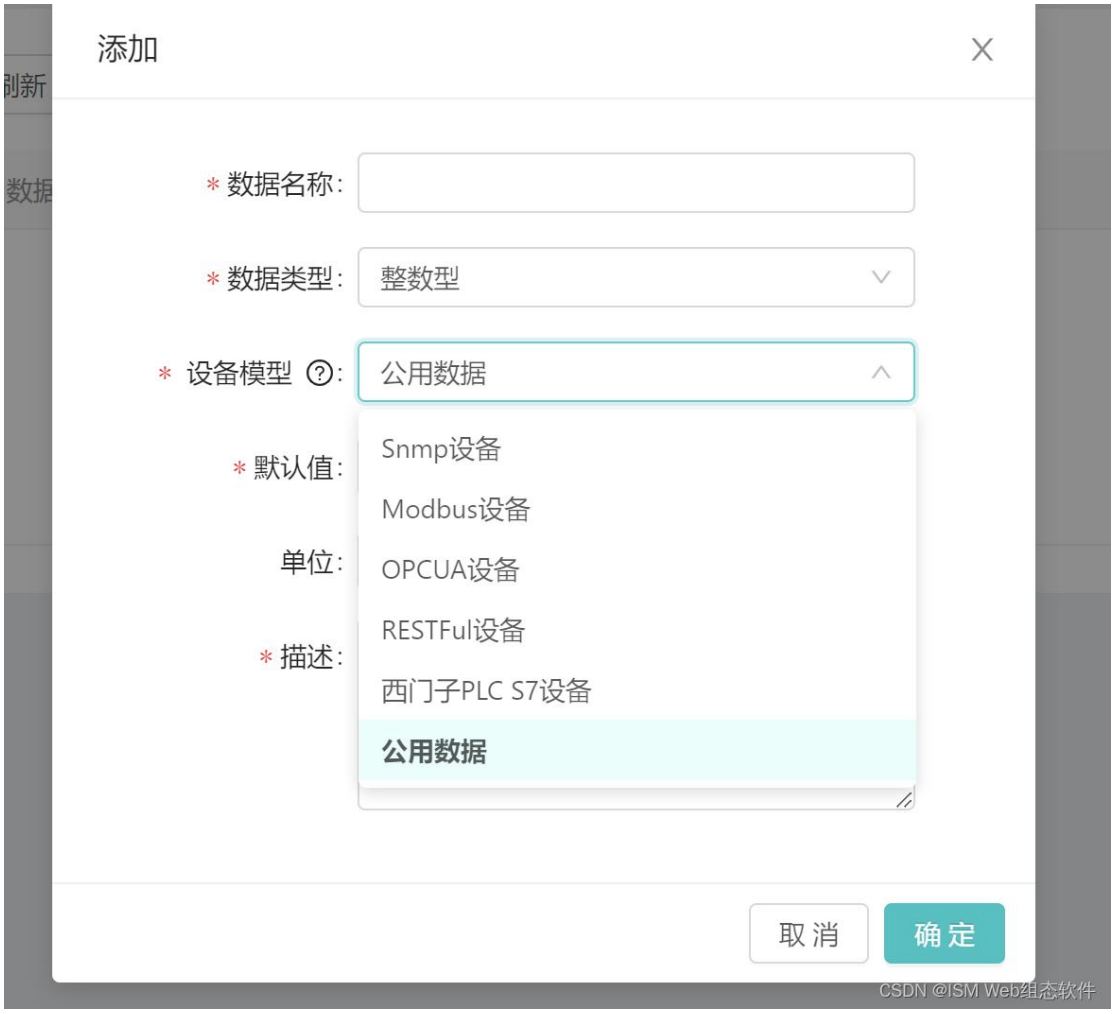
8.12.1 步骤 1

登陆 Hw-Link Web 组态软件，找到数据模型->系统变量，然后点击新建



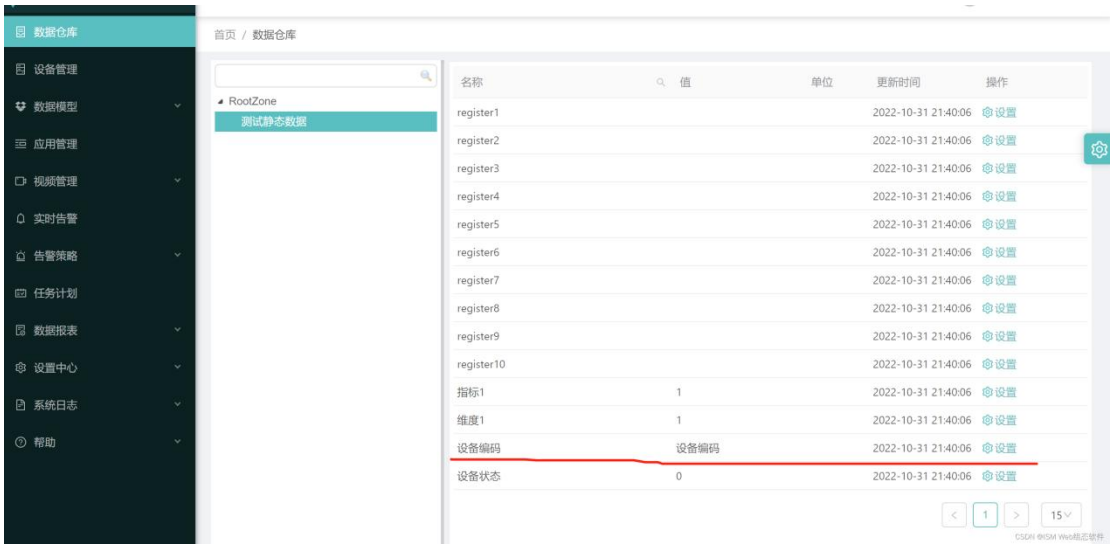
8.12.2 步骤 2

按照界面上的提示填写，这里有个设备模型，需要讲解一下，因为 Hw-Link 都是依据设备模型建立的设备，这里选择设备模型，是此设备属于那种设备模型，如果不选，默认是公用模型，所有的设备都会集成此数据。



8.12.3 步骤 3

然后在数据仓库中，可以看见所有的设备都会自动添加了此系统变量，这是可以设置，组态界面上可以显示此条数据。


数据仓库
设备管理
数据模型
应用管理
视频管理
实时告警
告警策略
任务计划
数据报表
设置中心
系统日志
帮助
首页 / 数据仓库
RootZone
测试静态数据
名称
值
单位
更新时间
操作
register1 2022-10-31 21:40:06 设置
register2 2022-10-31 21:40:06 设置
register3 2022-10-31 21:40:06 设置
register4 2022-10-31 21:40:06 设置
register5 2022-10-31 21:40:06 设置
register6 2022-10-31 21:40:06 设置
register7 2022-10-31 21:40:06 设置
register8 2022-10-31 21:40:06 设置
register9 2022-10-31 21:40:06 设置
register10 2022-10-31 21:40:06 设置
指标1 1 2022-10-31 21:40:06 设置
维度1 1 2022-10-31 21:40:06 设置
设备编码 设备编码 2022-10-31 21:40:06 设置
设备状态 0 2022-10-31 21:40:06 设置
1
15

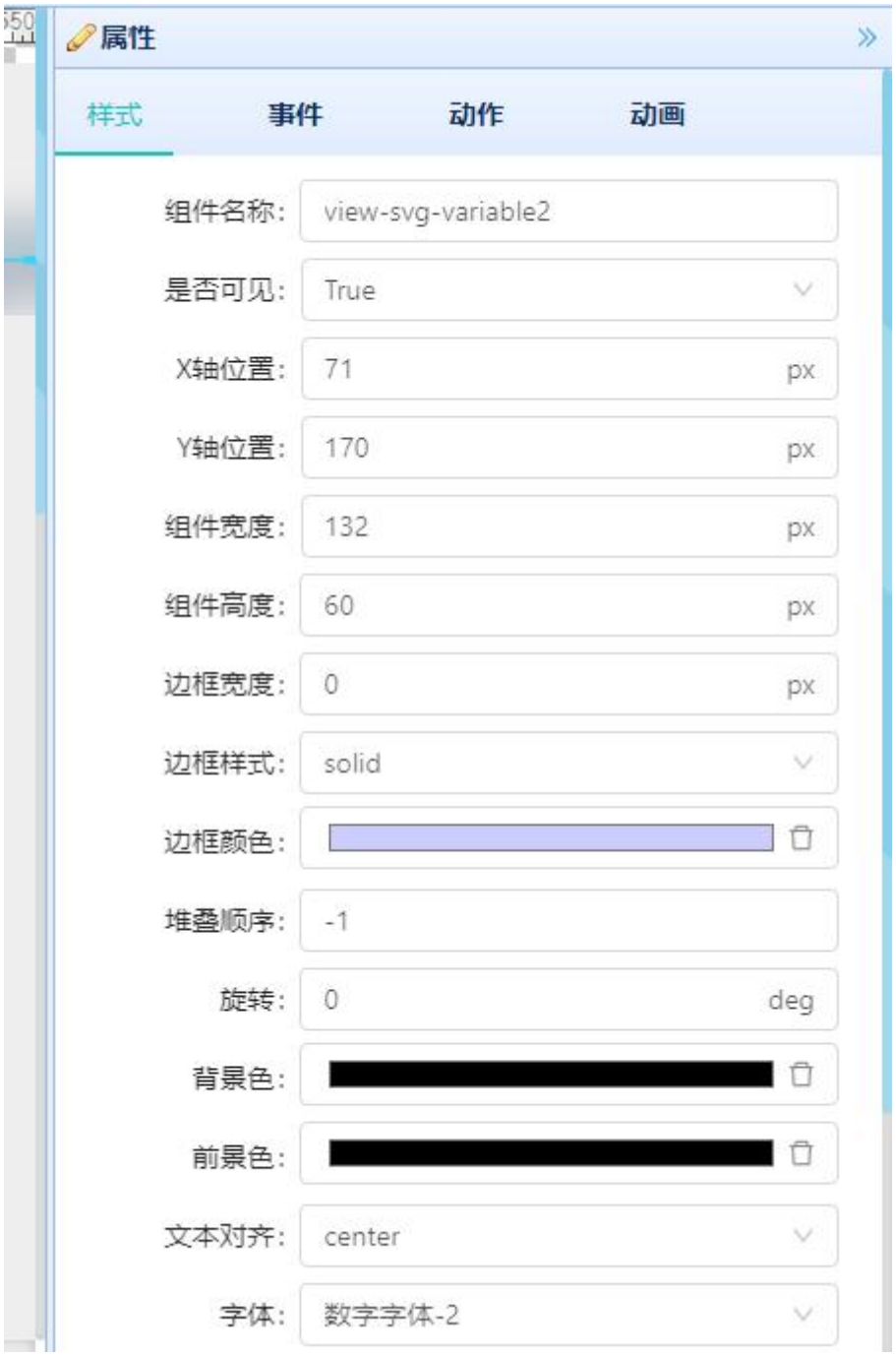
九、组态应用开发

9.1 简介

Hw-Link 的组态的每个组件都是从实际应用出发开发的，具有很强的实用性。组态界面分 PC 端和手机端两种界面，当用户在手机中输入 Hw-Link 地址时，会自动跳转到手机的界面展示

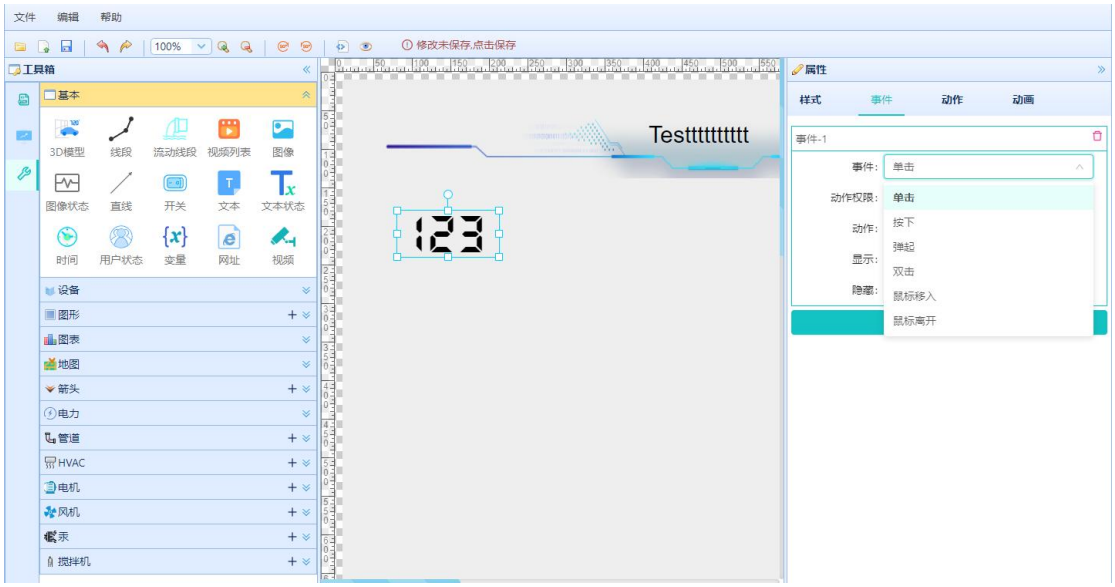
9.1.1 组件样式

每个组件都拥有基本的样式，这些样式规定了组件应该以什么样的样式呈现在界面上。
这里的堆叠顺序，可以理解为层，多个组件可以上下堆在一起



9.1.2 组件的事件

组件具有单击、按下、弹起、双击、鼠标移入、鼠标离开的事件，每个事件可以触发动作



打开网页:当事件产生时，Hw-Link 会主动打开配置好的网站或者内部的页面。

设置变量:条件满足时可以设置设备中的数据

显示或隐藏元素：条件满足时，可以显示界面上的某个组件，比如鼠标移入时，可以主动显示某个组件，用于显示当前的状态等。

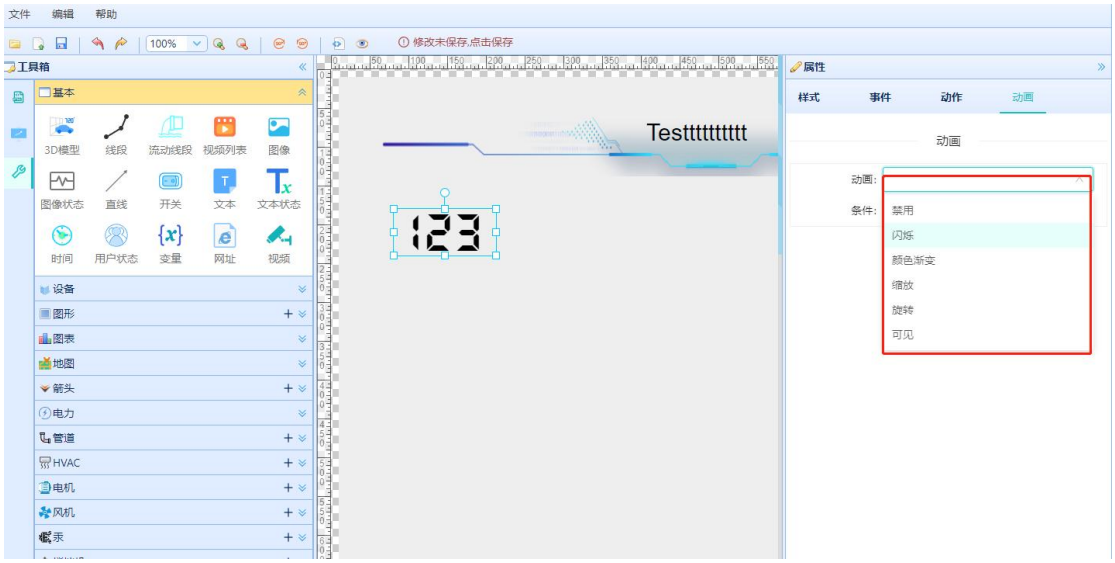
设备视图:这个可能不好理解，要慢慢理解一下，我举个例子，我有两种或更多种的设备，但是我不可能为每个设备都建立一个页面去展示，我只需要新建一个页面，就可以展示所有同类型的设备信息，这个设备视图就是可以跳转到数据模型建立的时候配置的页面。还记得在每个新建的数据模型中都会有个组态展示页面的配置吗？这个就是要跳转的页面。

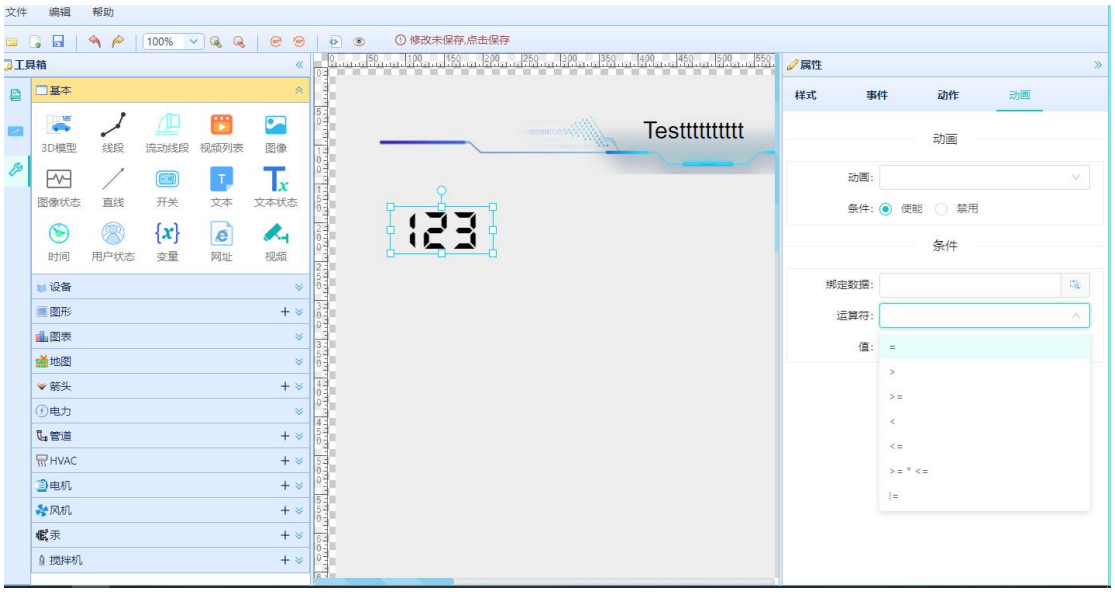
9.1.3 组件的动作

组件开发的时候我们都已经定义好，这个组件要显示怎样的数据，这里的动作就是要选择每个组件要显示的数据。

9.1.4 组件的动画

每个组件都可以播放动画，用户可以定义动画是否需要条件



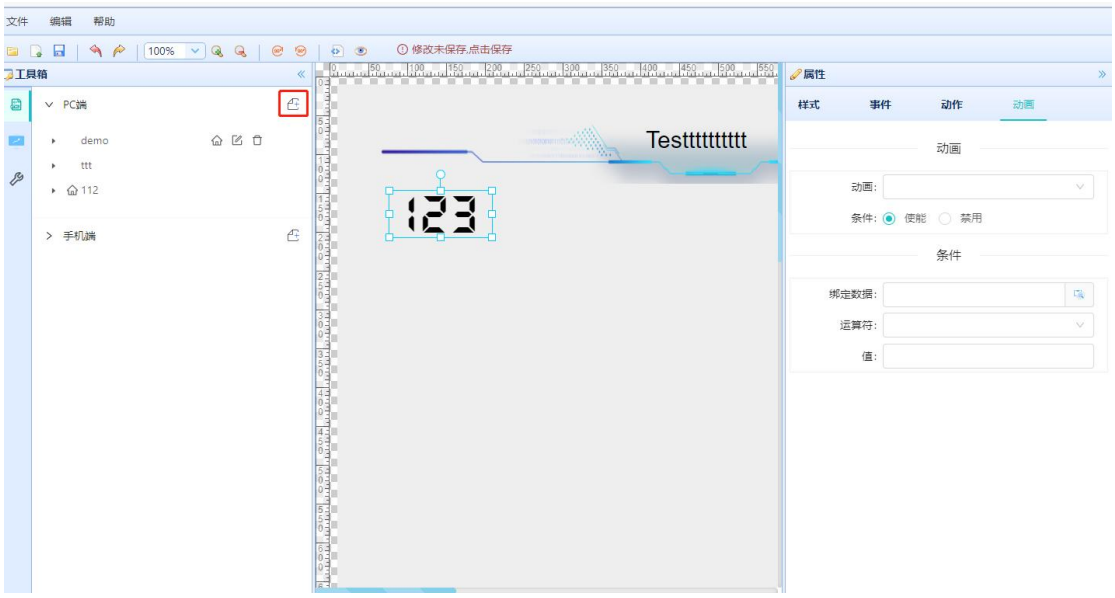


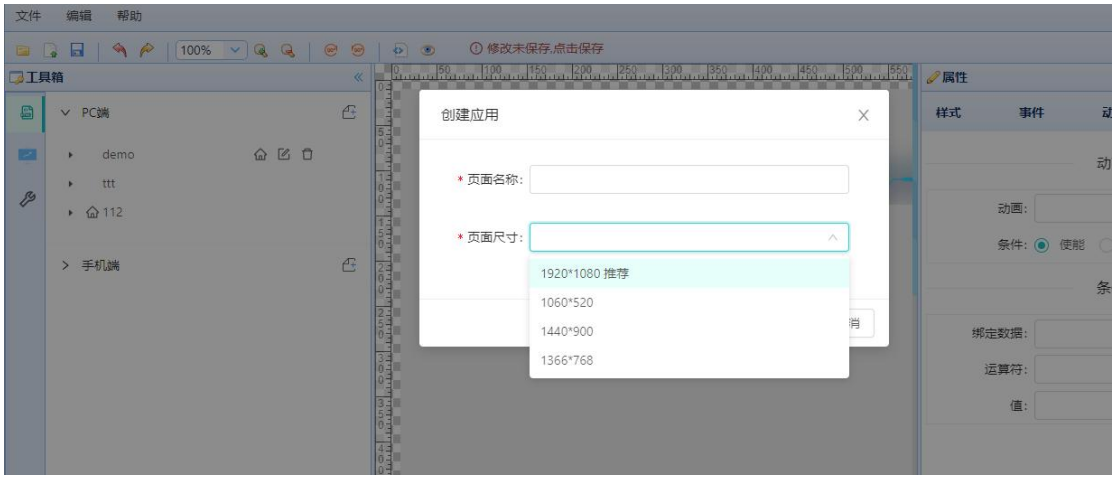
9.2 创建页面

页面是 Hw-Link 组态界面最重要的组成部分，数据的展示主要靠每个页面去展示，页面分电脑端和手机端，手机端页面在用户手机上打开时自动跳转到手机界面。手机界面也可以用 uni-app 工具直接打包成 android 和 IOS 平台的安装包。

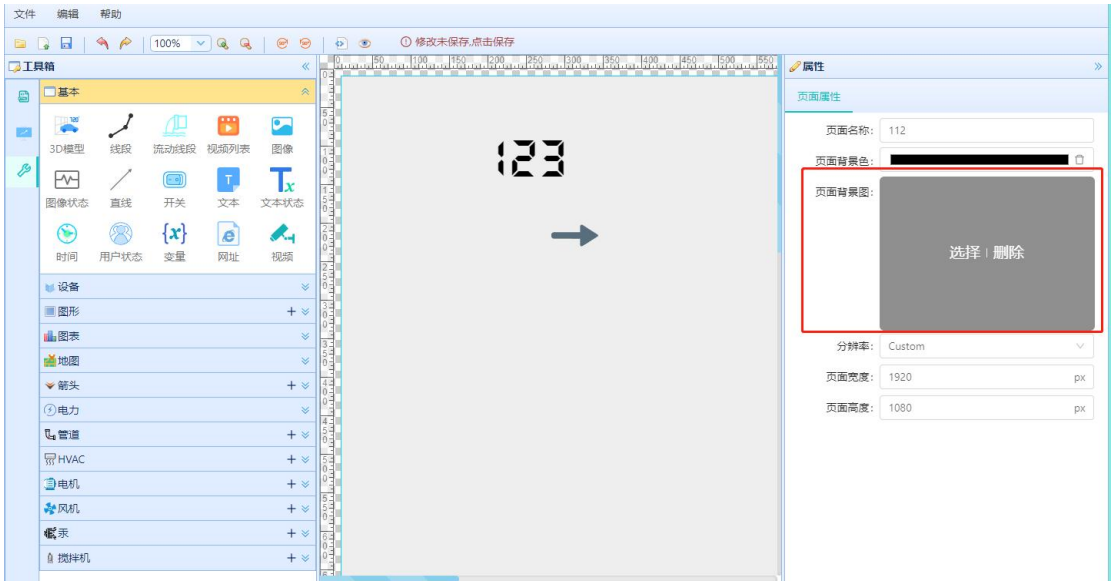
9.2.1 创建电脑端页面

点击 PC 端右边的图标，弹出新建页面的对话框，填写页面名称，选择页面尺寸，尺寸也可以在样式中修改

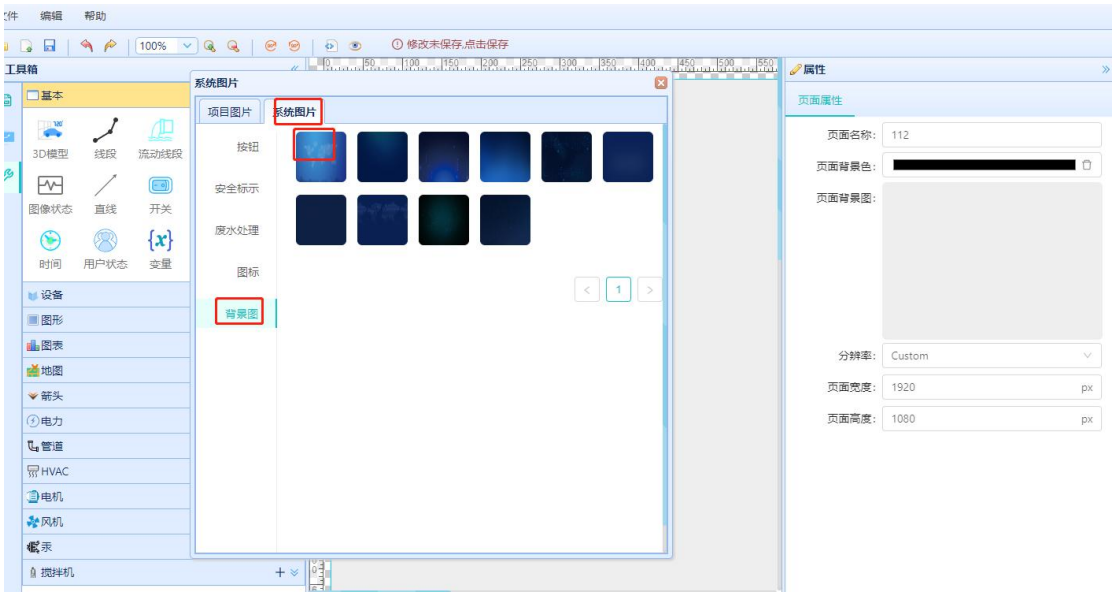




给页面加个背景图

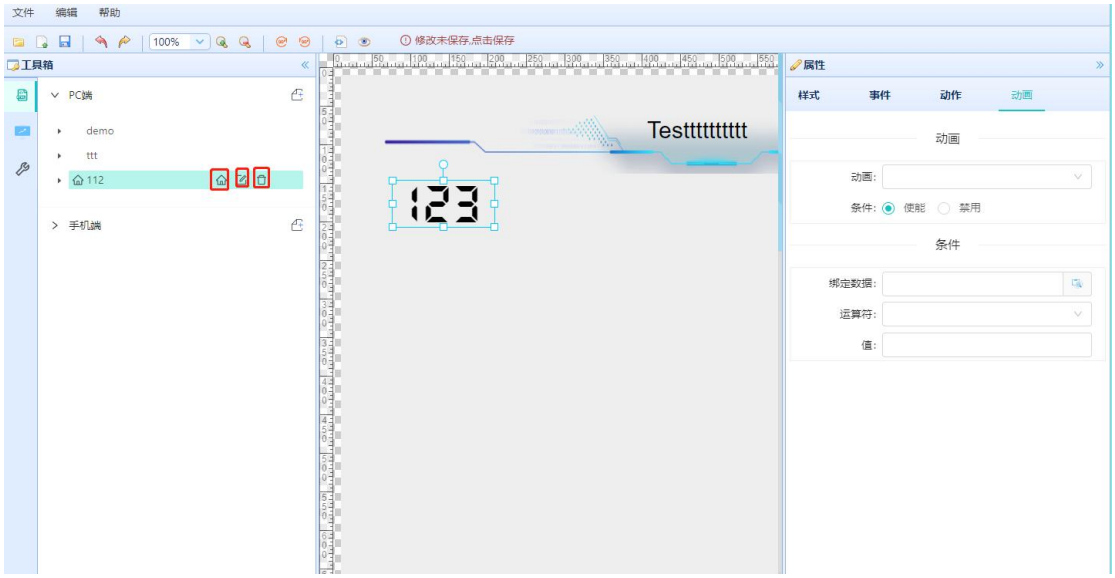


在系统图片中，选择一个背景图，然后页面的背景即可变成了我们想要的背景

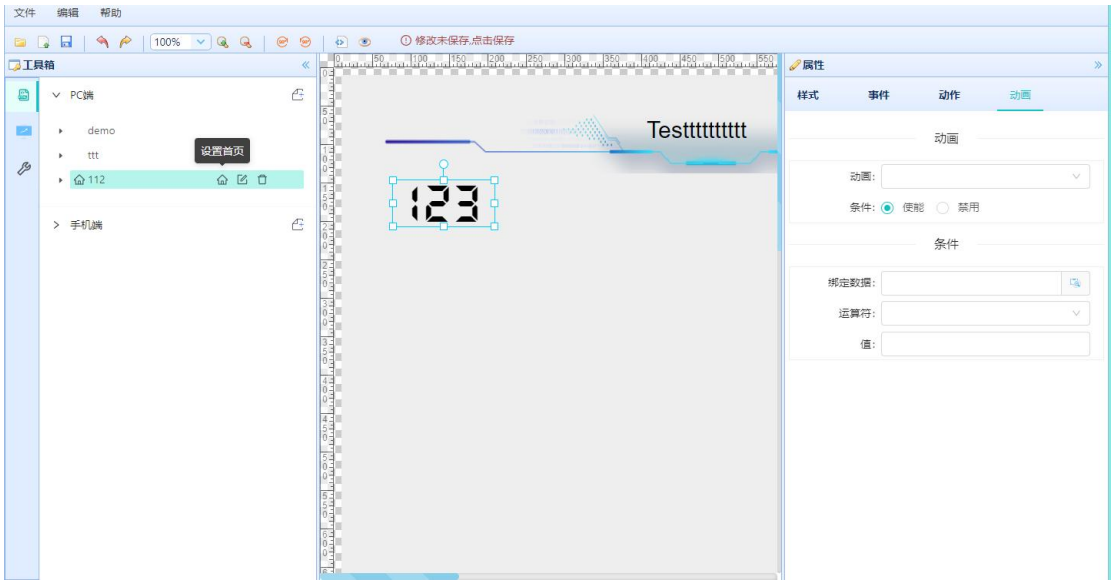


9.2.2 设置首页

如果我们有多个页面，那么我们必须要设置一个展示的首页，也就是打开应用时第一个展示的页面。如下图，点击像家的图标，即可设置本页面为此应用的首页展示，如果此页面是首页，页面的前面会出现像家的图标。

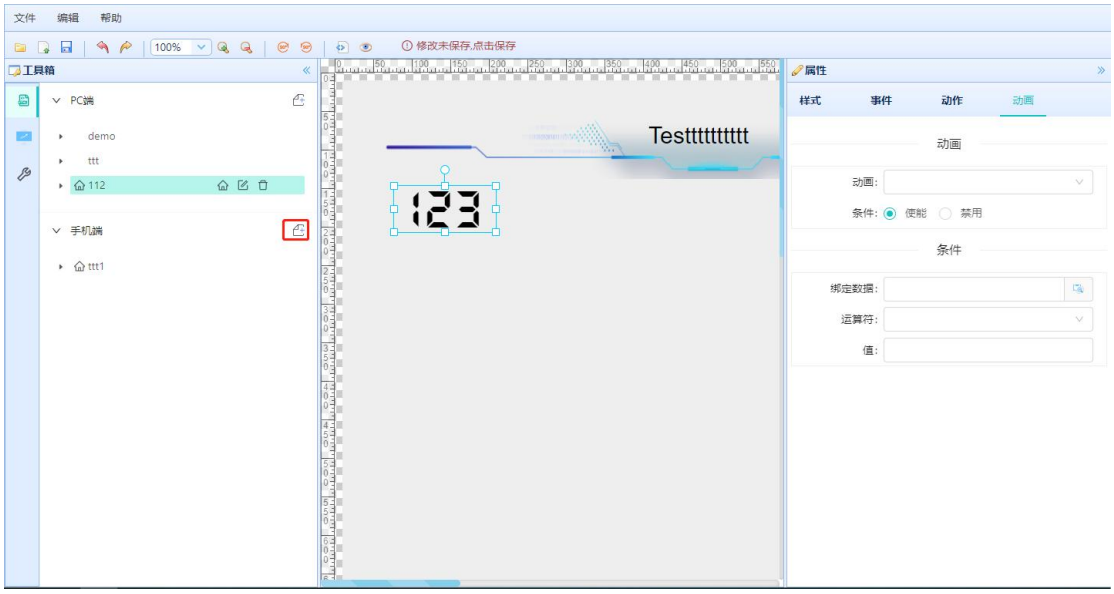


后面两个图标依次是，编辑页面的名称，删除页面，鼠标移动上面时会有提示操作。

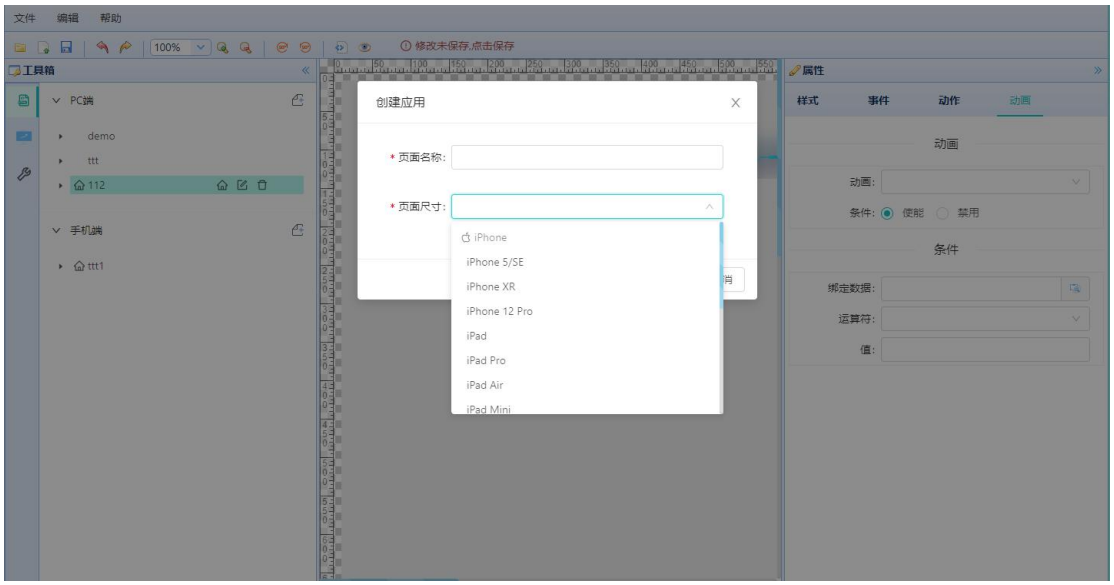


9.2.3 手机端页面

点击手机端右边的添加图标，



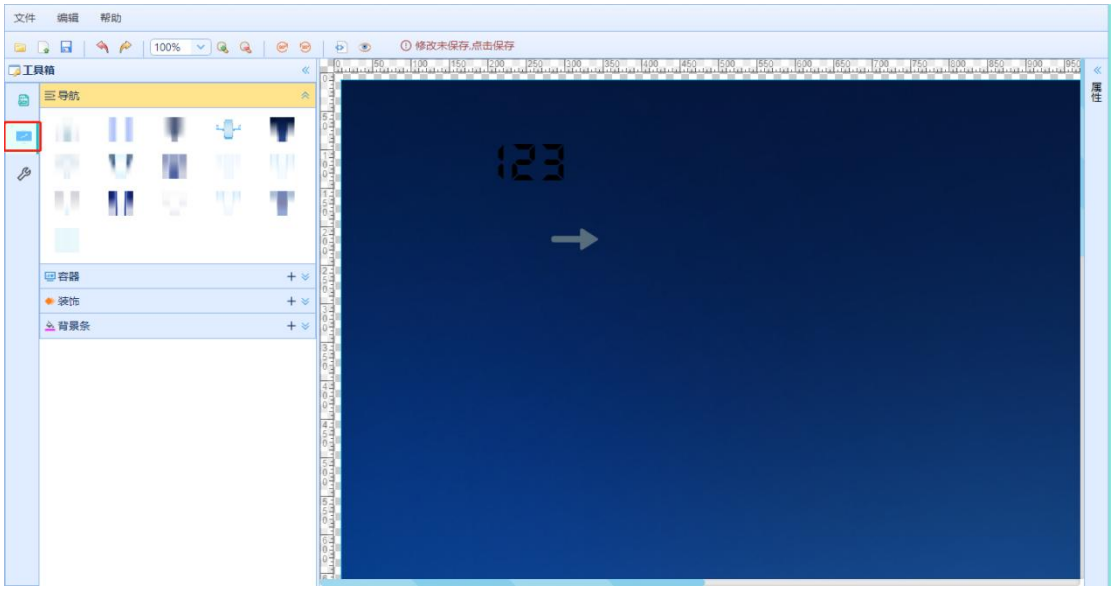
填写页面名称，和手机尺寸



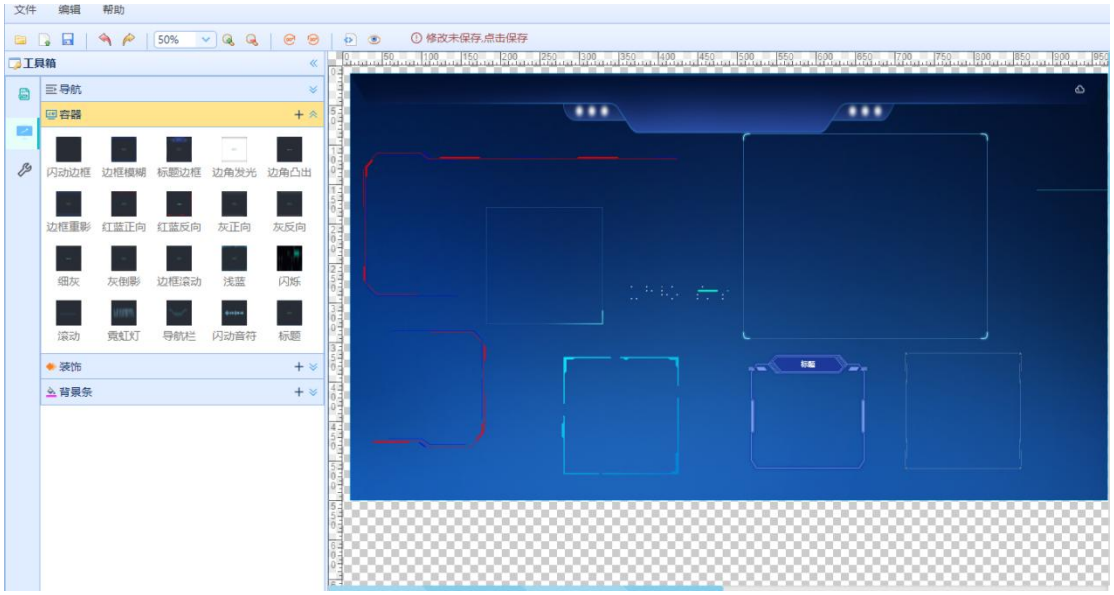
后面的同 PC 端页面操作，这里不再讲述。

9.2.4 装饰页面

Hw-Link 已经集成了很多好看的页面元素给用户使用，点击左侧的选项卡，这里列出来了很多页面需要的元素，比如导航条、背景容器、页面装饰条、还有很多的装饰条，点击+按钮，可以浏览很多。



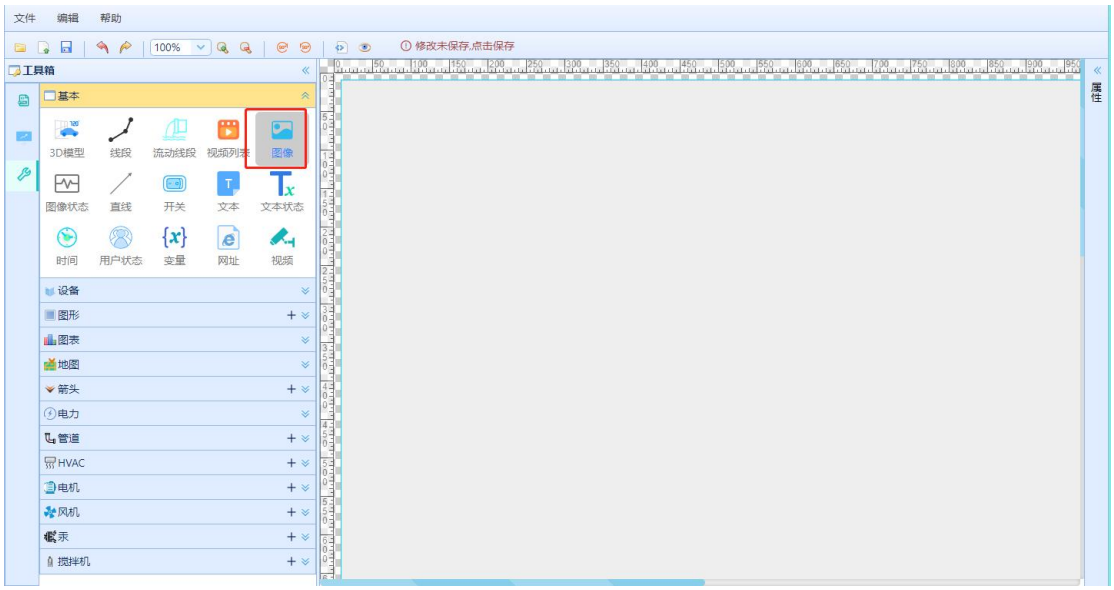
大家可以自己的想象力去装饰自己的页面。



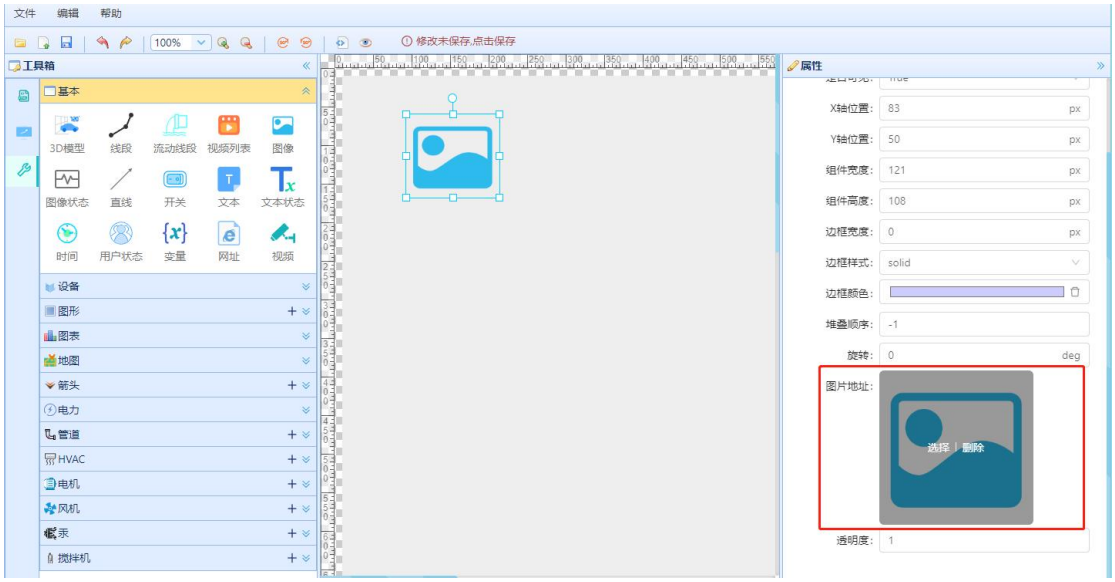
9.3 开发第一个组态应用

这节我们要学习怎么样在多个页面中进行页面的跳转。

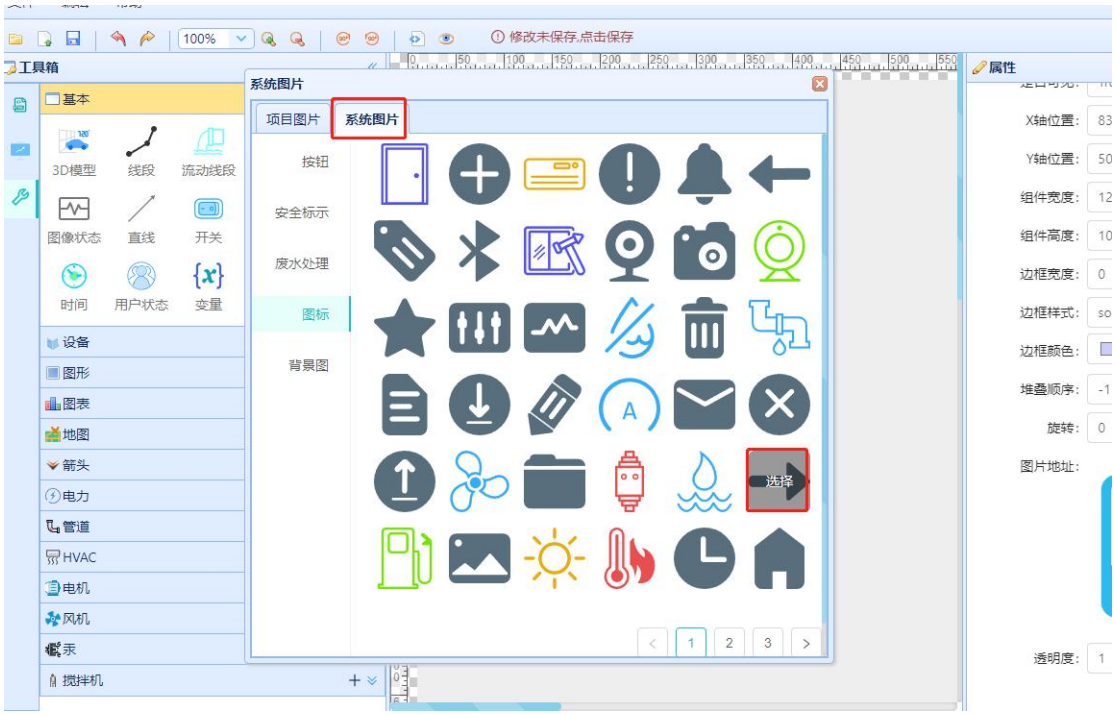
9.3.1 添加图片组件



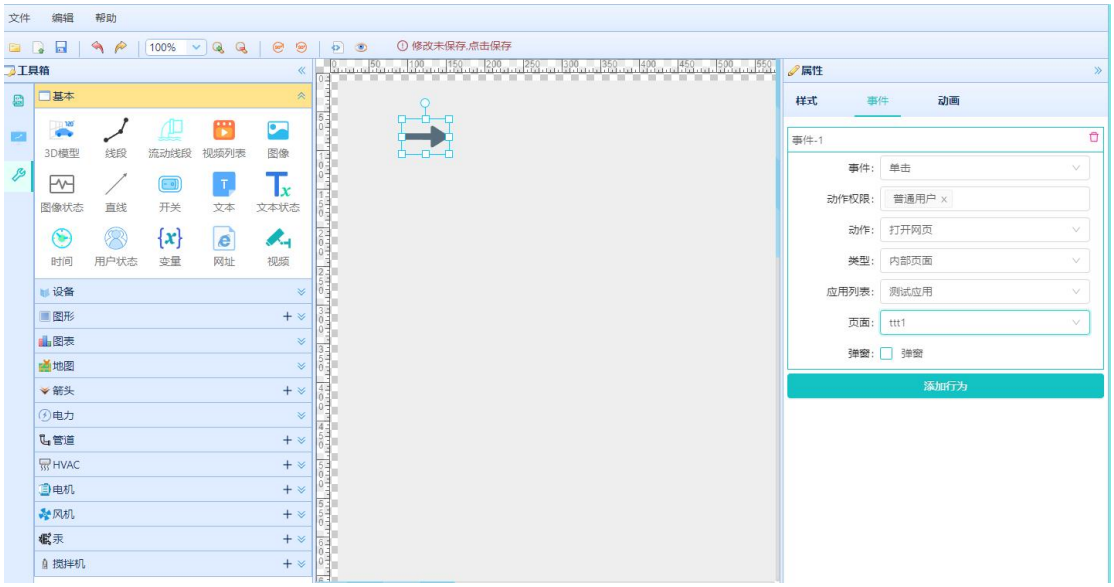
9.3.2 选择图像



9.3.3 选择系统图片，选择一个图标



选择我们图片组件，给组件添加一个单击行为，行为触发的的是打开应用里面的另一个页面。



Ctrl+s 即可保存，点击预览按钮调试我们刚才开发的页面

但我们的鼠标移动到组件上时，显示当前组件是可以点击的，点击后跳转到了我们刚才配置的页面中

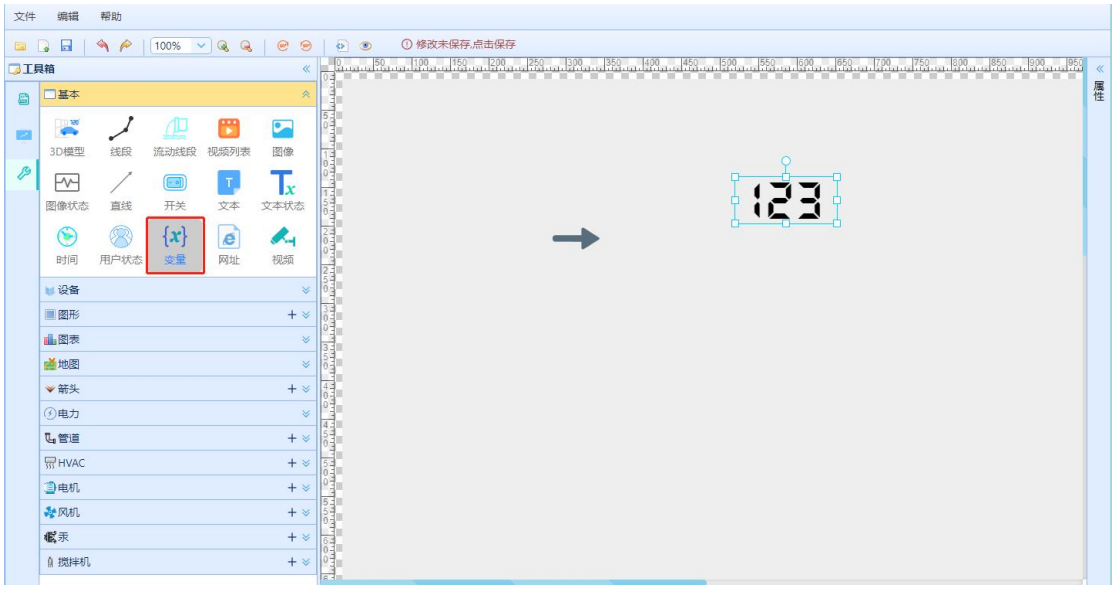
行为里面页面打开有个弹窗选项，此选项是不进行跳转，直接以弹窗的形式打开页面

9.4 展示设备的数据

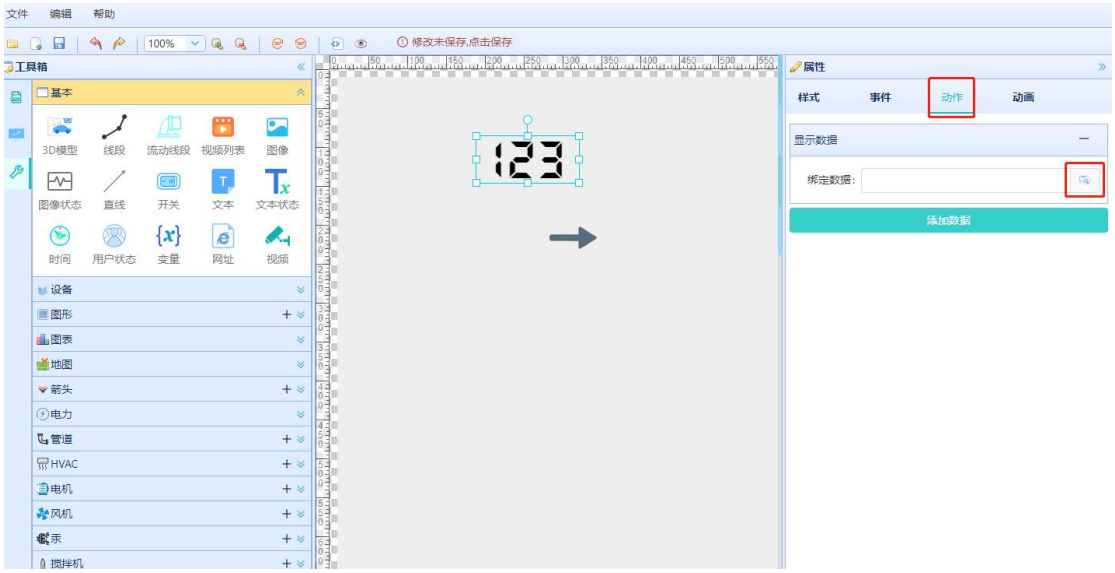
这节我们将展示如果让组态界面与我们的设备联动，前提是我们的设备数据已经采集已经完成。

9.4.1 展示设备变量

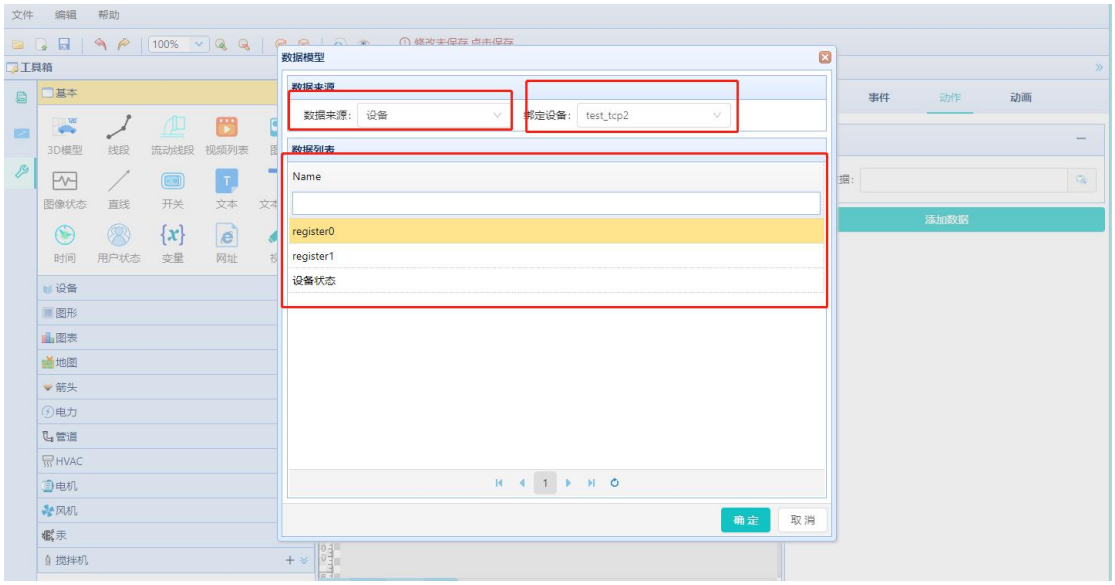
在左侧列表中把变量组件拖到页面中，选择我们要显示的设备数据



点击动作页面，然后选择绑定的数据



在弹窗的对话框中选择我们要展示的数据



完成后，ctrl+s 保存，单击预览按钮，即可看到设备的数据同步显示到页面中

9.5 开发自定义登录界面

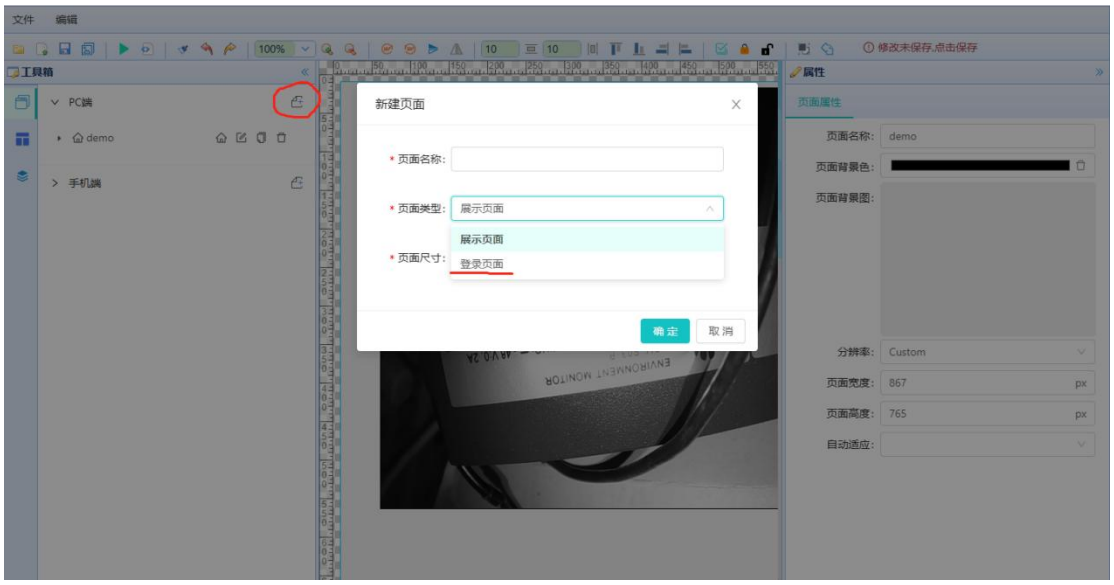
有时我们需要做很酷炫的登录界面，替换掉 Hw-Link 原来的登录界面，这时我们就可以通过组态界面自定义登录界面。

注意

此功能需要购买授权后使用

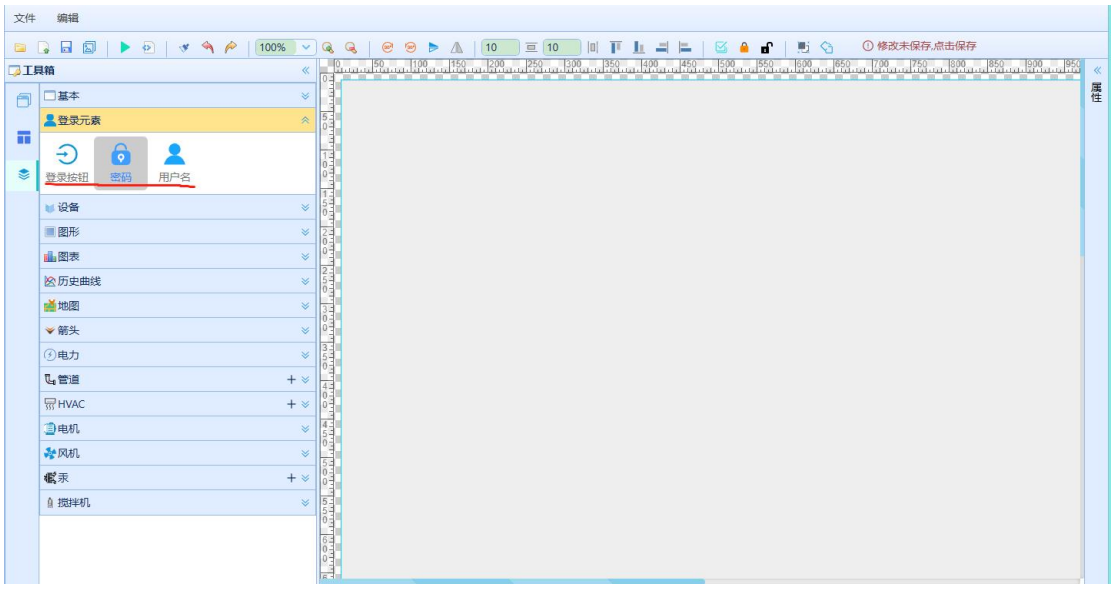
9.5.1 添加登录页面

点击添加页面按钮，页面类型选择登录界面



9.5.2 添加登录元素

点击登录元素，把三个元素拖进页面中，根据实际需要制作登录页面

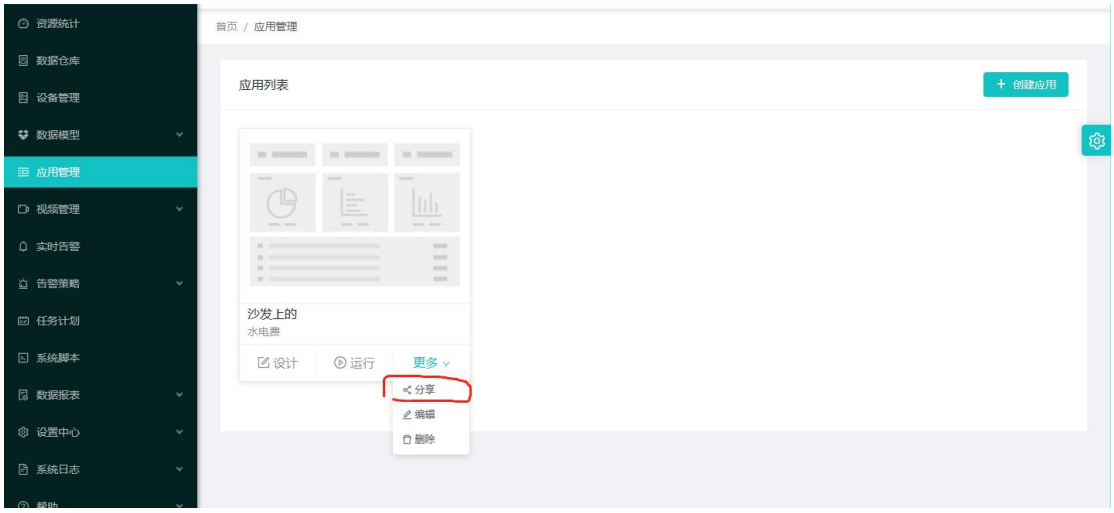


9.5.3 测试登录页面

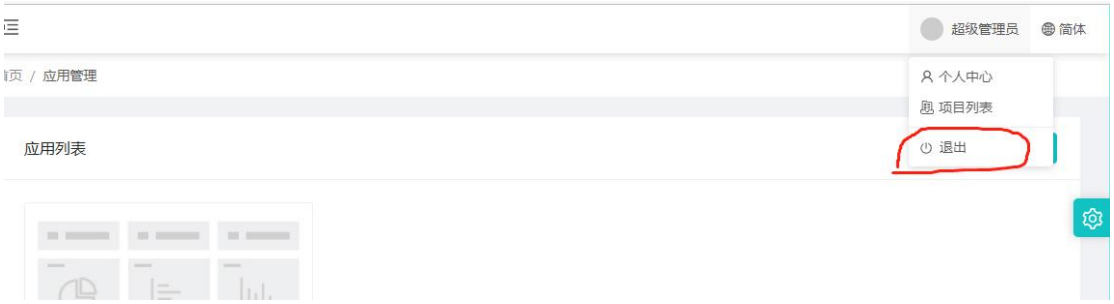
复制当前的 APP 的运行地址，比如

http://127.0.0.1:8081/#/AppRun/5f6495fb-624d-1551-4285-a4609683092c

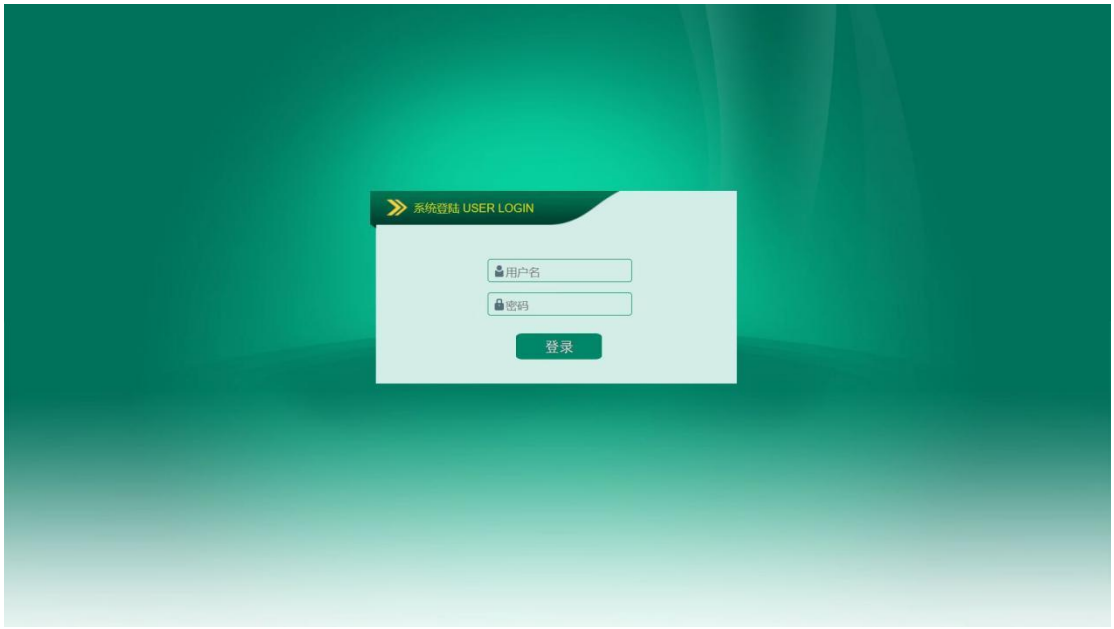
如果是局域网访问，请把 127.0.0.1 地址改为你局域网的 IP 地址。



退出当前登录的账号，http://127.0.0.1:8081/#/AppRun/5f6495fb-624d-1551-4285-a4609683092c



在浏览器里面输入你要分享的 APP 运行地址，即可测试登录界面



9.6 免登录组态界面分享

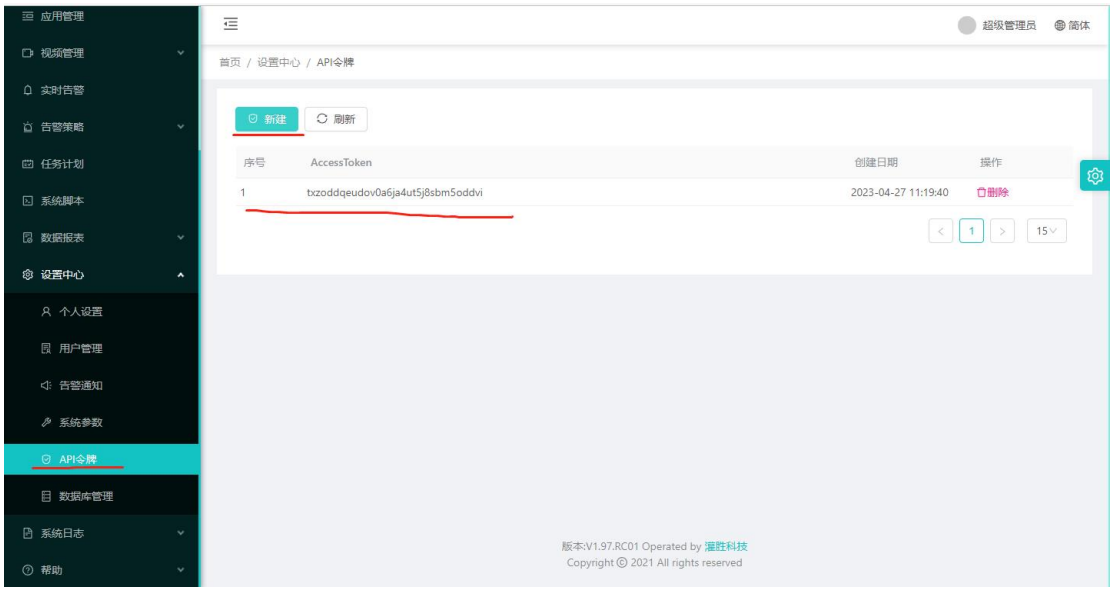
Hw-Link 为了把制作好的组态页面，嵌入到第三方系统中，特意开发了免登录组态运行地址。方便嵌入第三方网页中。

注意

此功能需要购买授权后使用

9.6.1 申请 Token 的令牌

点击设置中心的，API 令牌。点击新建，系统会自动生成一个 token



9.6.2 嵌入地址 URL 说明

http://127.0.0.1:8081/#/ShareApp/5f6495fb-624d-1551-4285-a4609683092c/txzoddqeudov0a6ja4ut5j8sbm5oddvi

127.0.0.1:8081 是 Hw-Link 运行的网络地址，可以更改为 Hw-Link 运行的外网地址。

5f6495fb-624d-1551-4285-a4609683092c 是 APP 的 ID

txzoddqeudov0a6ja4ut5j8sbm5oddvi 是免登录的 token

注意

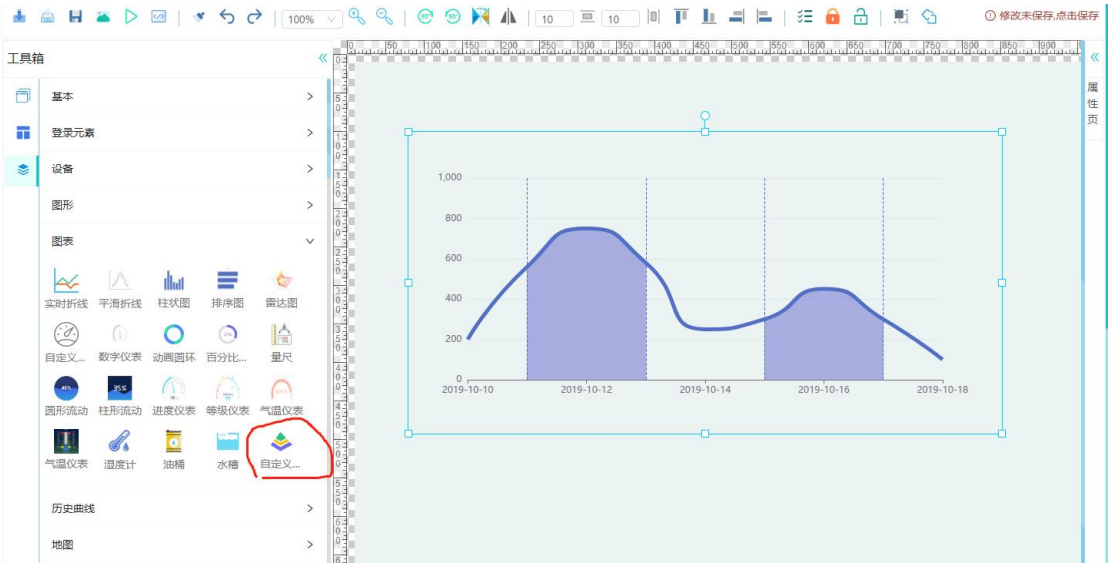
注意：路径里是 ShareApp 而不是 AppRun

9. 7 自定义图表

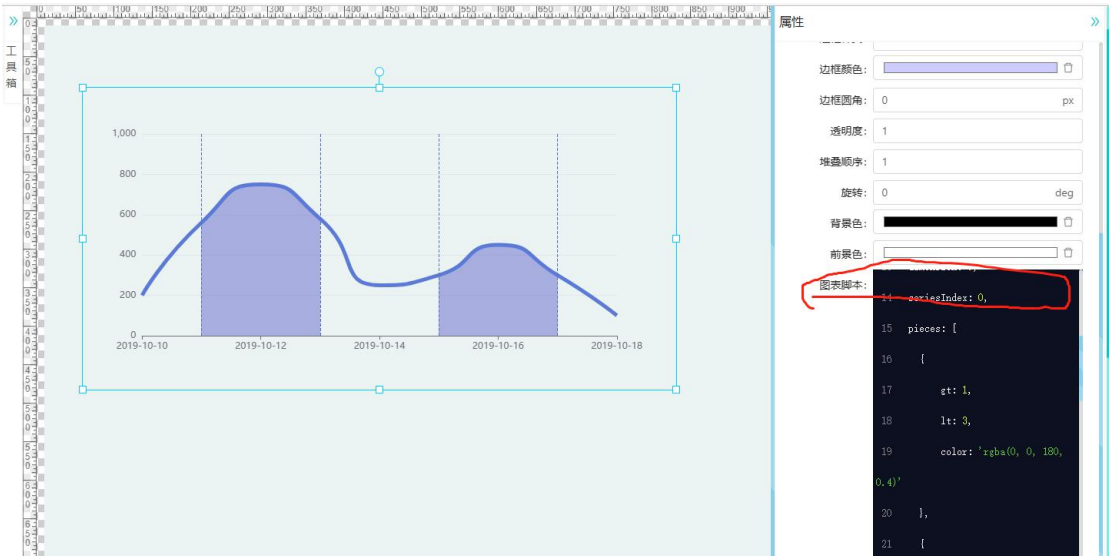
为了满足更多的用户对图表的需要，Hw-Link 特意增加了自定义图表组件，用户可以根据自己的需要自己动手开发需要的展示图表。

9.7.1 组态画面拖入自定义图表

在左侧列表中把自定义图表组件拖到页面中



点击刚刚拖入的自定义图表组件，点击属性页，找到图表脚本选项



双击图表脚本内容，弹出编辑脚本内容的对话框。用户可以对话框里面添加自己想要展示的图表

9.7.2 变量和函数说明

this.echartsView 是 echarts 初始化完成后的变量，用户可以调用 echarts 官网上的函数

this.echartsView.setOption(option,true);这里就是把 echarts 的图表选项重新绘制

注意：在定时器中不能使用 this 去指向，用户需要通过定义全局变量去指向此值

```
var thatEchartsView = this.echartsView;
thatEchartsView.setOption(option,true);
```

ComponentRestApi 函数是请求远程 APi 的函数

第一个参数是 Get 或者 Post

第二个参数是请求地址

第三个参数是请求的参数 json 对象

例子：

```
setInterval(function(){
```

```
ComponentRestApi("Get","https://echarts.apache.org/examples/data/asset/data/life-expectancy-table.json",{}).
then(function (res){
                console.log(res)
}).catch(function(){

})
}, 1000);
```

完整例子：


```
 let value = 0.2;
 let data = [value, value, value, ];
let option = {
  color: ['#80FFA5', '#00DDFF', '#37A2FF', '#FF0087', '#FFBF00'],
  title: {
    text: 'Gradient Stacked Area Chart'
  },
  tooltip: {
    trigger: 'axis',
    axisPointer: {
      type: 'cross',
      label: {
        backgroundColor: '#6a7985'
      }
    }
  },
  legend: {
    data: ['Line 1', 'Line 2', 'Line 3', 'Line 4', 'Line 5']
  },
  toolbox: {
    feature: {
      saveAsImage: {}
    }
```

```
},
grid: {
  left: '3%',
  right: '4%',
  bottom: '3%',
  containLabel: true
},
xAxis: [
  {
    type: 'category',
    boundaryGap: false,
    data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
  }
],
yAxis: [
  {
    type: 'value'
  }
],
series: [
  {
    name: 'Line 1',
    type: 'line',
    stack: 'Total',
    smooth: true,
    lineStyle: {
      width: 0
    },
    showSymbol: false,
    areaStyle: {
      opacity: 0.8,
      color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
        {
          offset: 0,
          color: 'rgb(128, 255, 165)'
        },
        {
          offset: 1,
          color: 'rgb(1, 191, 236)'
        }
      ])
    },
    emphasis: {
      focus: 'series'
```

```
    },
    data: [140, 232, 101, 264, 90, 340, 250]
  },
  {
    name: 'Line 2',
    type: 'line',
    stack: 'Total',
    smooth: true,
    lineStyle: {
      width: 0
    },
    showSymbol: false,
    areaStyle: {
      opacity: 0.8,
      color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
        {
          offset: 0,
          color: 'rgb(0, 221, 255)'
        },
        {
          offset: 1,
          color: 'rgb(77, 119, 255)'
        }
      ])
    },
    emphasis: {
      focus: 'series'
    },
    data: [120, 282, 111, 234, 220, 340, 310]
  },
  {
    name: 'Line 3',
    type: 'line',
    stack: 'Total',
    smooth: true,
    lineStyle: {
      width: 0
    },
    showSymbol: false,
    areaStyle: {
      opacity: 0.8,
      color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
        {
          offset: 0,
```

```
                color: 'rgb(55, 162, 255)'
              },
              {
                offset: 1,
                color: 'rgb(116, 21, 219)'
              }
            ])
        },
        emphasis: {
          focus: 'series'
        },
        data: [320, 132, 201, 334, 190, 130, 220]
      },
      {
        name: 'Line 4',
        type: 'line',
        stack: 'Total',
        smooth: true,
        lineStyle: {
          width: 0
        },
        showSymbol: false,
        areaStyle: {
          opacity: 0.8,
          color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
            {
              offset: 0,
              color: 'rgb(255, 0, 135)'
            },
            {
              offset: 1,
              color: 'rgb(135, 0, 157)'
            }
          ])
        },
        emphasis: {
          focus: 'series'
        },
        data: [220, 402, 231, 134, 190, 230, 120]
      },
      {
        name: 'Line 5',
        type: 'line',
        stack: 'Total',
```

```
        smooth: true,
        lineStyle: {
          width: 0
        },
        showSymbol: false,
        label: {
          show: true,
          position: 'top'
        },
        areaStyle: {
          opacity: 0.8,
          color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
            {
              offset: 0,
              color: 'rgb(255, 191, 0)'
            },
            {
              offset: 1,
              color: 'rgb(224, 62, 76)'
            }
          ])
        },
        emphasis: {
          focus: 'series'
        },
        data: [220, 302, 181, 234, 210, 290, 150]
      }
    ]
};
let thatChartView = this.echartsView
thatChartView.setOption(option,true);

setInterval(function(){
  option.series[0].data = [220, 302, 181, 234, 210, 290, 150]
  thatChartView.setOption(option,true);
}, 1000);
```

9.8 简述

组态界面上的很多组件大家可以自己去探究，因为每个组件没有什么逻辑，都是很好理解的，我再这里就不再一一讲述了，有什么不明白的地方可咨询客服。

十、视频采集

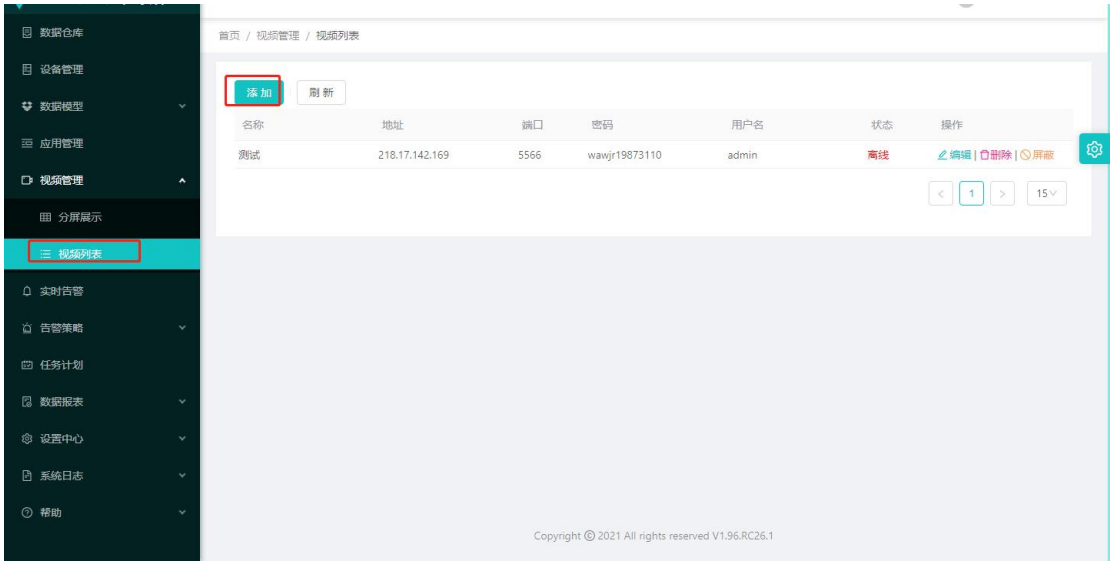
10.1 视频添加

相关信息

本节讲述视频的添加

10.1.1 添加视频

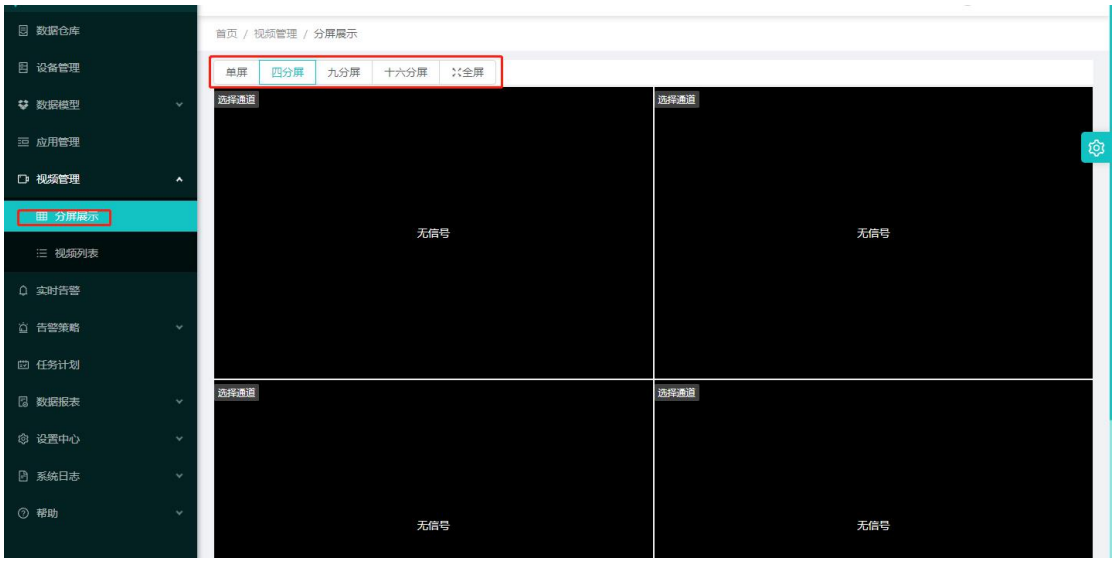
登录 Hw-Link，依次点击视频列表->添加



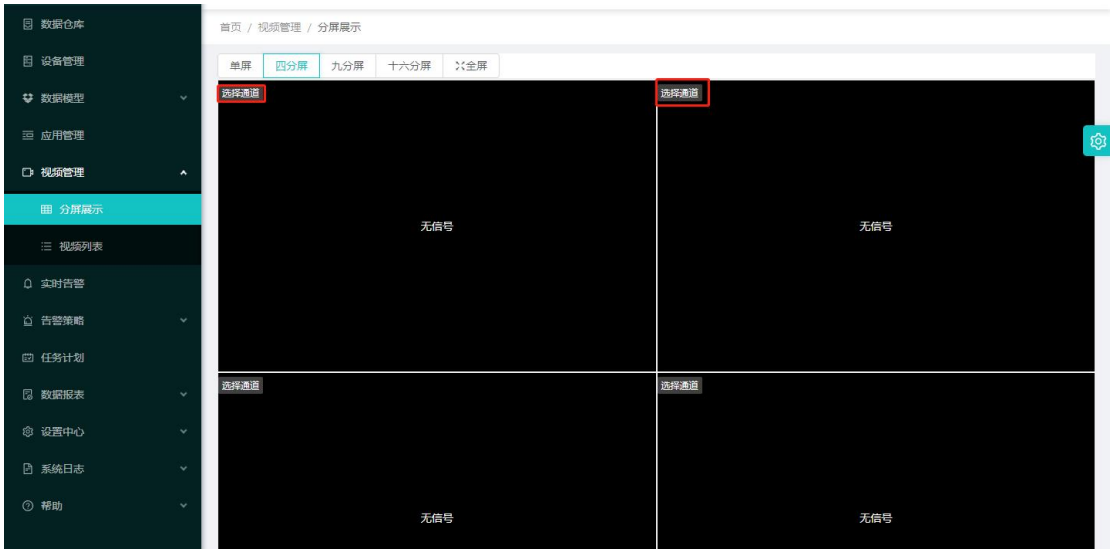
地址填摄像机的 IP 地址，端口一般是 554，RTSP 地址这是根据每个摄像机设定的。下面会给出主流摄像机的地址，这里的地址只需要填写 RTSP 后面的地址，不需要完整的地址。Hw-Link 会自己拼接地址。

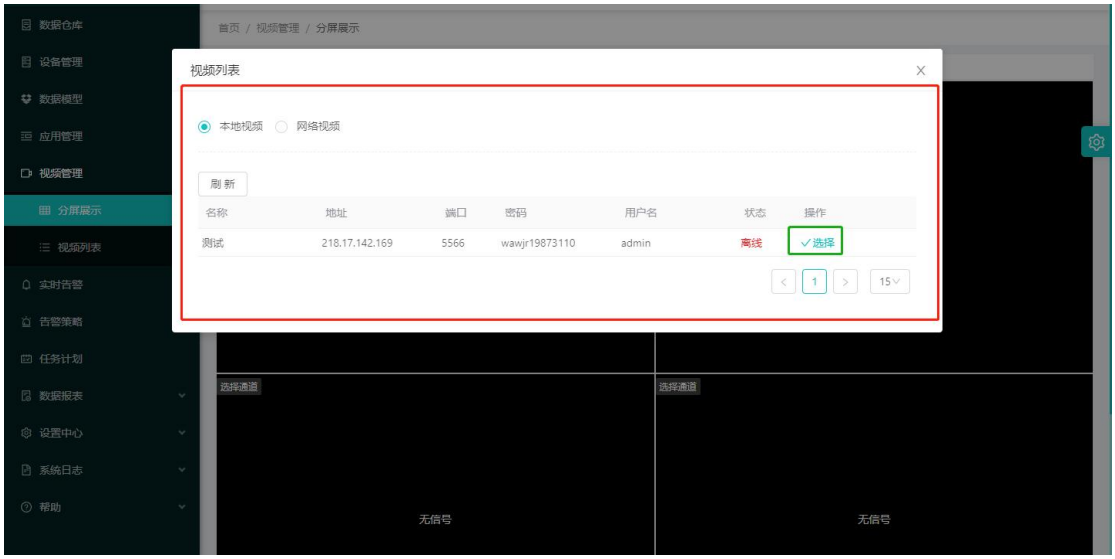
10.2 视频查看

登录 Hw-Link，点击分屏展示，这里可以选择分屏的数量



点击选择通道，在弹窗的选项框中选择我们要查看的摄像机





10.3 视频录像

相关信息

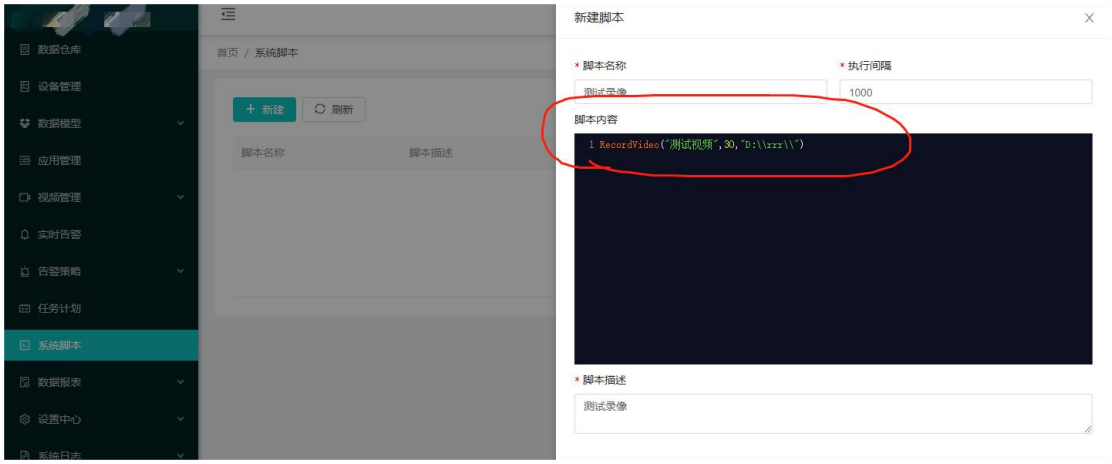
本节讲述视频录像的使用

在系统脚本里面，RecordVideo 函数

第一个参数：视频名称(这个是添加的视频名称)

第二个参数：录像的时间，单位为秒

第三个参数：保存路径 注意 windows 下面的路径有反斜杠，要这样表达两个反斜杠表达一个反斜杠



10.4 SRS 视频服务器播放 GB28181 视频流

相关信息

SRS 是一个开源的（MIT 协议）简单高效的实时视频服务器，支持 RTMP、WebRTC、HLS、HTTP-FLV、SRT、MPEG-DASH 和 GB28181 等协议。 SRS 媒体服务器和 FFmpeg、OBS、VLC、 WebRTC 等客户端配合使用，提供流的接收和分发的能力，是一个典型的发布 （推流）和订阅（播放）服务器模型。 SRS 支持互联网广泛应用的音视频协议转换，比如可以将 RTMP 或 SRT， 转成 HLS 或 HTTP-FLV 或 WebRTC 等协议。

10.4.1 安装

```
./configure --gb28181=on
make
./objs/srs -c conf/gb28181.conf
```

然后，在摄像头配置中，选择 AAC 编码，然后在平台中配置 SIP 服务器为 SRS，如下图所示：

必须是 AAC 编码，在音频编码中，选择 AAC，采样率 44100HZ。

必须是 GB-2016 标准，否则不支持 TCP，在协议版本中选择 GB/T28181-2016。

必须是 TCP 协议，不支持 UDP，在传输协议中选择 TCP，并使用 GB-2016 标准。

摄像头注册后，SRS 会自动邀请摄像头推流，可以打开下面的链接播放：

http://localhost:8080/live/34020000001320000001.flv

http://localhost:8080/live/34020000001320000001.m3u8

webrtc://localhost/live/34020000001320000001

Note: 请把流名称换成你的设备名称，然后点播放。

详细的步骤，请参考下面的网址

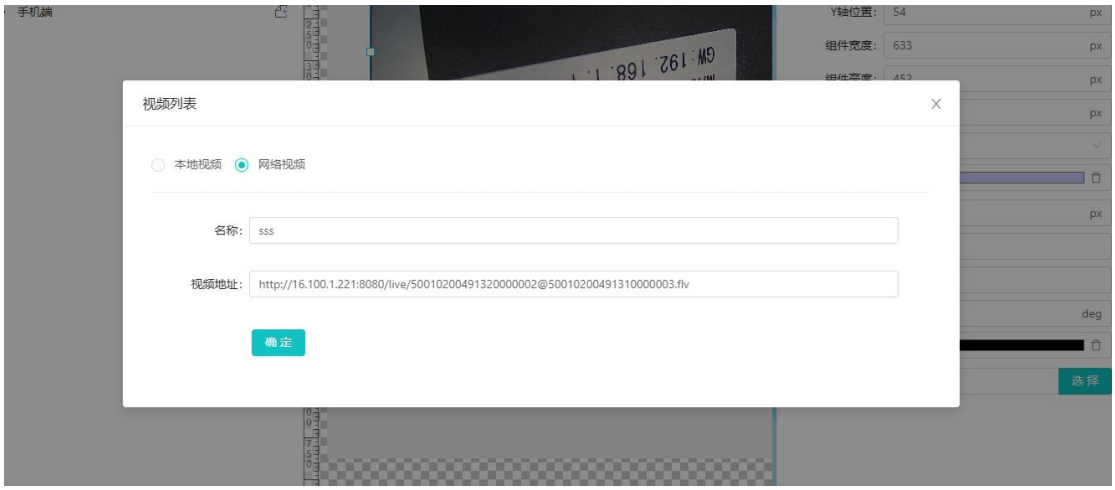
https://ossrs.net/lts/zh-cn/docs/v5/doc/gb28181

10.4.2 获取视频地址

登录 SRS 的网址，找到视频流，然后找到你添加的摄像机，点击预览

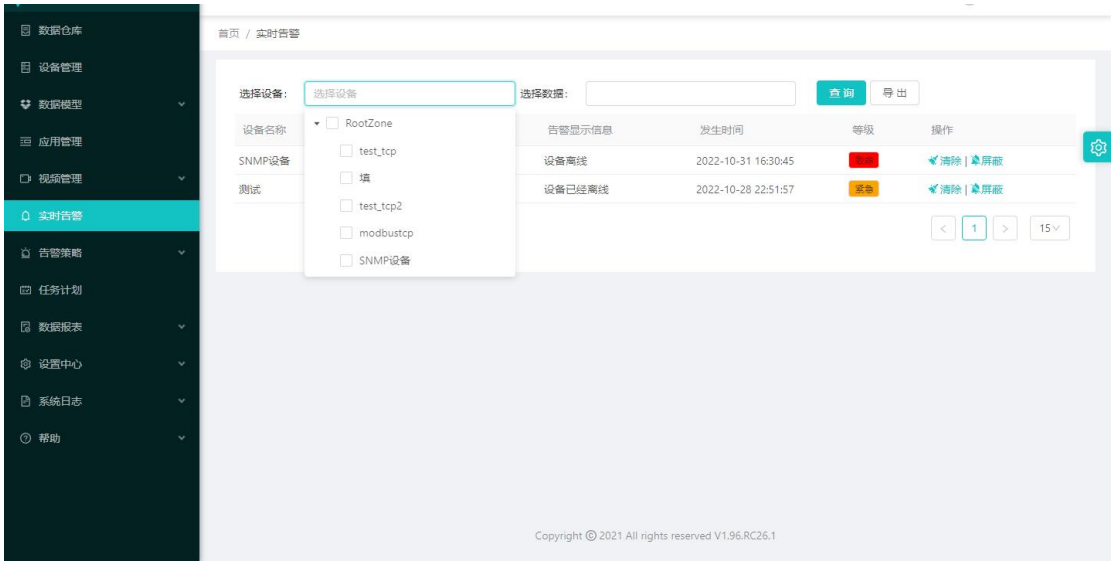
复制播放的 URL 地址，在 Hw-Link 的视频组件里面的网络组件添加网络地址，把复制的地址拷贝进去


SRS
SRS播放器
RTC播放器
RTC推流
iOS/Andriod
GB28181
GitHub Repo stars
Usage: 输入HTTP-FLV/HLS地址后点击"播放视频"即可播放视频
URL: http://16.100.1.221:8080/live/500102004913200000002@500102004913100000003.flv
播放视频
2023年08月11日 星期五 13:03:22
分享: 请右键拷贝此链接


手机端
视频列表
本地视频
网络视频
名称: sss
视频地址: http://16.100.1.221:8080/live/500102004913200000002@500102004913100000003.flv
确 定
Y轴位置: 54
组件宽度: 633
选 择

十一、实时告警

实时告警可以查看当前项目中的设备的告警信息，可以根据设备和告警名称查询



清除按钮:

清除告警后，如果再次发生，Hw-Link 还会主动推送告警，前台也会显示

屏蔽按钮：

屏蔽后，告警消除，后面再发生告警，Hw-Link 不会主动推送告警，前台也不会显示

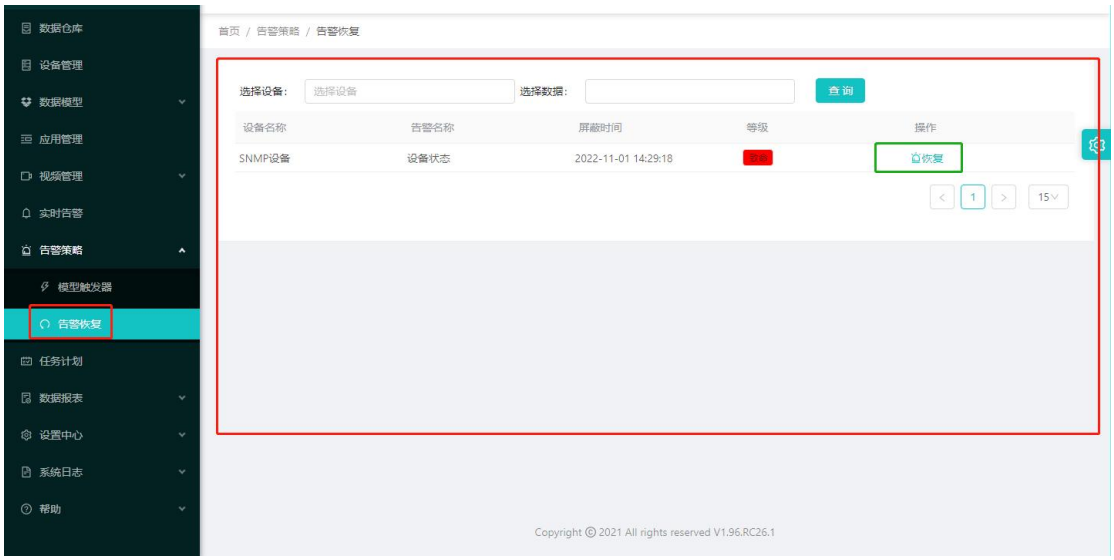
十二、告警策略

12.1 简介

告警策略分模型触发器和告警恢复
告警策略可以按条件触发特定的事件，还可以联动其他的数据
告警恢复是针对屏蔽后的告警，可以重新恢复告警

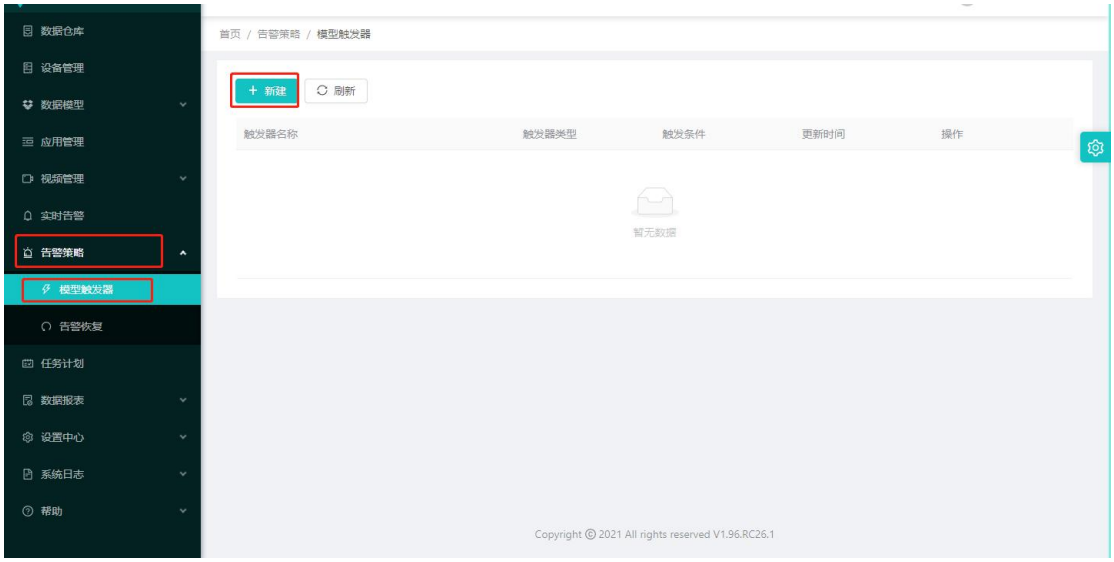
12.2 告警恢复

在列出来的告警屏蔽中，点击恢复按钮既可以恢复屏蔽的告警。



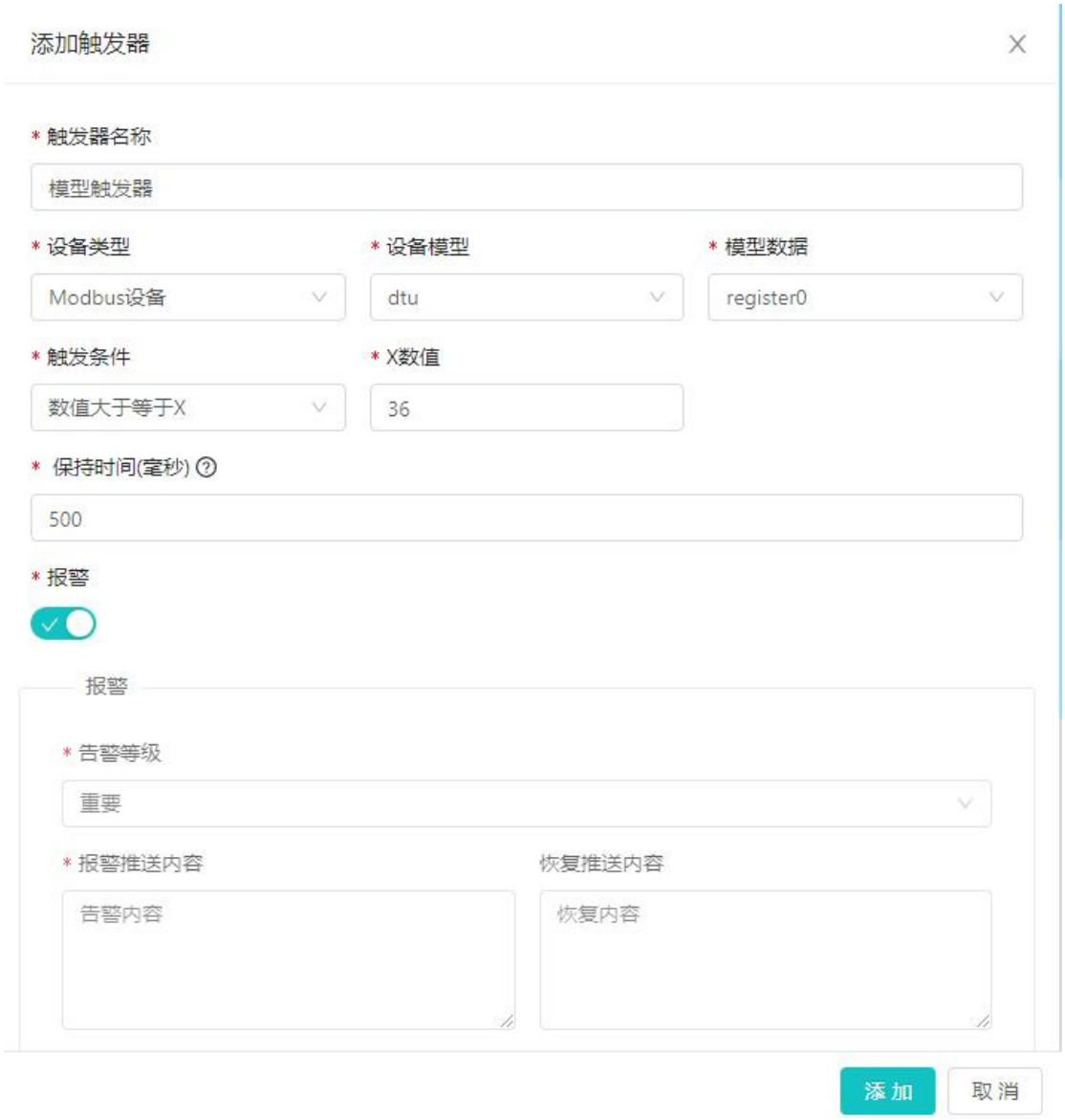
12.3 模型触发器

依次点击告警策略->模型触发器->新建



在弹出的页面中选择设备类型->设备模型->模型数据

设置好触发条件，条件值，保持时间等，Hw-Link 会根据条件自动计算，当设备类型中设备满足条件时即可推送告警内容，如果条件消除，Hw-Link 也会推送消除内容，如果消除内容为空，Hw-Link 不会再推送。



告警的同时也可以联动其他的模型数据，分为告警时下发的数据和消除时下发的数据。具体参见联动界面


* 联动数据
联动数据
* 联动模型数据
register1
* 联动告警下发数据值
10
* 联动告警清除下发数据值
15

十三、GO 虚拟机

13.1 简介

为了满足灵活的逻辑和复杂的业务需求，Hw-Link 特增加了 Go 虚拟机功能，用户可以根据场景的实际情况，写程序去满足实际的业务需要。

Hw-Link 提供了的虚拟机，类似于 Go 语言语法，大家可以去像编写 Go 语言一样编写 Hw-Link 的提供的程序。

Hw-Link 启动时会从 sys_script 目录中读取程序文件，并运行。

注意：当程序修改后需要重启 Hw-Link 才能生效。

13.2 简单的逻辑和运算例子

```
# 定义函数
func foo(x){
   return x + 1
}

func bar(x ...){
   return len(x)
}

# 定义变量
x = 1
y = x + 1

# 打印变量
println(x * (y + 2 * x + foo(x) / 2))

# if/else condition
if foo(y) >= 1 {
   println("你好，Hw-Link")
} else {
   println("www.hwzdh.cn")
}

# 定义数组
```

```
a = [1,2,3]
println(a)
println(a[2])
println(len(a))

# 定义 map
m = {"foo": "bar", "bar": "baz"}
for k in keys(m) {
    println(m[k])
}

f = func(a) {
    println(a)
}


f = func(a ...) {
    println(a)
}


println(1 && 2)

println(bar(1,2,3))
println("foo")
println(toByteSlice("ISM 切片"))
println(toRuneSlice("ISM 切片"))

a = 1
func foo() {
    a = 2
}
foo()

println(a)

module Foo {
    func bar1() {
        println("Foo.bar1")
    }
}

println(Foo.bar1())
```

13.3 获取操作系统类型

```
var os, runtime = import("os"), import("runtime")

if runtime.GOOS == "windows" {
   println(os.Getenv("USERPROFILE"))
} else {
   println(os.Getenv("HOME"))
}
```

13.4 执行系统的程序

```
var os, exec = import("os"), import("os/exec")

cmd = exec.Command("ls", "-la")
cmd.Stdout = os.Stdout
cmd.Run()
```

13.5 for 循环

例子 1:

```
func fib(n) {
   a, b = 1, 1
   f = []
   for i in range(n) {
      f += a
      b += a
      a = b - a
   }
   return f
}


println(fib(20))
```

例子 2:

```
func fib(n) {
   if n == 1 {
      return [1]
```

```
    } else if n == 2 {
        return [1,1]
    } else {
        t = fib(n-1)
        return t + (t[len(t)-1] + t[len(t)-2])
    }
}

println(fib(20))
```

例子 3:

```
for i in [1,2,3,4,5] {
    if i == 2 {
        continue
    }
    println(i)
    if i > 3 {
        break
    }
    println("foo")
}
```

13.6 Http 远程调用

```
//简单的 Get 请求，具体查询 net/http 包的 Get 和 Post 请求
var http, ioutil = import("net/http"), import("io/ioutil")

r = http.DefaultClient.Get("http://golang.org/")
b, _ = ioutil.ReadAll(r[0].Body)
printf("%s", toString(b))
r[0].Body.Close()
```

```
//增加头部参数的请求
var http, ioutil = import("net/http"), import("io/ioutil")

  var apiEndpoint="http://127.0.0.1:8099/DiyChartData"
  var req,w=http.NewRequest("GET", apiEndpoint, nil)
  Print(req)
  req.Header.Add("Content-Type","application/json")
  req.Header.Add("Authorization","Bearer d9965291-42f0-4dde-8960-256063166b97")
var lient = make(http.Client)
```

```
var resp, err = lient.Do(req)
Print(resp)
```

13.7 Go 模块使用

```
module Foo {
    func bar1() {
        println("Foo.bar1")
        return 1
    }
}

println(Foo.bar1())
```

13.8 Go Http 服务

```
var http = import("net/http")

http.HandleFunc("/", func(w, r) {
    w.Write(toByteSlice("hello world"))
})

http.ListenAndServe(":8099", nil)
```

13.9 Go 类型转换

```
# 转为整型
println("转整型例子")
i = 1<<63 - 1
println("int", i, "toInt:", toInt(i))

i = -1 << 63
println("int", i, "toInt:", toInt(i))

f = 3.141592653589793
println("float", f, "toInt:", toInt(f))

f = 1.797693134862315708145274237317043567981e18
println("float", f, "toInt:", toInt(f))

f = -1.797693134862315708145274237317043567981e18
println("float", f, "toInt:", toInt(f))
```

```
s = "4611686018427387904"
println("string", s, "toInt:", toInt(s))

s = "-9223372036854775808"
println("string", s, "toInt:", toInt(s))

s = "3.141592653589793"
println("string", s, "toInt:", toInt(s))

s = "1.797693134862315708145274237317043567981e18"
println("string", s, "toInt:", toInt(s))

s = "-1.797693134862315708145274237317043567981e18"
println("string", s, "toInt:", toInt(s))

s = "1.797693134862315708145274237317043567981e-18"
println("string", s, "toInt:", toInt(s))

b = true
println("bool", b, "toInt:", toInt(b))

b = false
println("bool", b, "toInt:", toInt(b))

println("转浮点数")
i = 1<<63 - 1
println("int", i, "toFloat:", toFloat(i))

i = -1 << 63
println("int", i, "toFloat:", toFloat(i))

s = "4611686018427387904"
println("string", s, "toFloat:", toFloat(s))

s = "-9223372036854775808"
println("string", s, "toFloat:", toFloat(s))

s = "3.141592653589793"
println("string", s, "toFloat:", toFloat(s))

s = "1.797693134862315708145274237317043567981e18"
println("string", s, "toFloat:", toFloat(s))
```

```
s = "-1.797693134862315708145274237317043567981e18"
println("string", s, "toFloat:", toFloat(s))

s = "1.797693134862315708145274237317043567981e-18"
println("string", s, "toFloat:", toFloat(s))

b = true
println("bool", b, "toFloat:", toFloat(b))

b = false
println("bool", b, "toFloat:", toFloat(b))

println("\n 转 bool")
i = 1
println("int", i, "toBool:", toBool(i))

i = 0
println("int", i, "toBool:", toBool(i))

i = -1
println("int", i, "toBool:", toBool(i))

f = 1.0
println("float", f, "toBool:", toBool(f))

f = 0.000000000001
println("float", f, "toBool:", toBool(f))

f = 0.0
println("float", f, "toBool:", toBool(f))

f = -0.0
println("float", f, "toBool:", toBool(f))

s = "y"
println("string", s, "toBool:", toBool(s))

s = "yEs"
println("string", s, "toBool:", toBool(s))

s = "t"
println("string", s, "toBool:", toBool(s))

s = "TrUe"
```

```
println("string", s, "toBool:", toBool(s))

s = "1"
println("string", s, "toBool:", toBool(s))

s = "0"
println("string", s, "toBool:", toBool(s))

s = "f"
println("string", s, "toBool:", toBool(s))

s = "FaLsE"
println("string", s, "toBool:", toBool(s))

s = "foobar"
println("string", s, "toBool:", toBool(s))
```

13.10 Go 串口操作例子

```
var serial = import("serial")
var time = import("time")
var fmt = import("fmt")
var bytes = import("bytes")
var io = import("io")
var binary = import("encoding/binary")
var strconv = import("strconv")

var serialConfig = new(serial.Config)
serialConfig.Address="COM1"
serialConfig.BaudRate=9600
serialConfig.DataBits=1
serialConfig.Parity="N"
serialConfig.StopBits=1
serialConfig.Timeout=10 * time.Millisecond

sendBuffer = make([]byte, 16)
sendBuffer[0]=0x68
sendBuffer[1]=0x01
sendBuffer[2]=0x00
sendBuffer[3]=0x00
sendBuffer[4]=0x00
sendBuffer[5]=0x00
sendBuffer[6]=0x00
sendBuffer[7]=0x68
```

```
sendBuffer[8]=0x11
sendBuffer[9]=0x04
sendBuffer[10]=0x33
sendBuffer[11]=0x33
sendBuffer[12]=0x34
sendBuffer[13]=0x33
sendBuffer[14]=0xB3
sendBuffer[15]=0x16

var port,err = serial.Open(serialConfig)

readByte = make([]byte, 30)

BeforeCode = make([]byte, 4)
DLT645Address = make([]byte, 6)
DLT645Start = make([]byte, 1)
ControlType = make([]byte, 1)
DLT645DataLen = make([]byte, 1)

DLT645DataCode = make([]byte, 4)
DLT645Data = make([]byte, 4)
DLT645DataBlack = make([]byte, 1)
DLT645Data1 = make([]byte, 4)

for{
    port.Write(sendBuffer)
    time.Sleep(1000*time.Millisecond)


    readByteSize, _ = io.ReadFull(port, readByte)
    fmt.Println(readByteSize)
    fmt.Println(readByte[:readByteSize])


    bufIO = bytes.NewReader(readByte)

    binary.Read(bufIO, binary.LittleEndian, BeforeCode)
    fmt.Println(BeforeCode)

    binary.Read(bufIO, binary.LittleEndian, DLT645Start)
    fmt.Println(DLT645Start)

    binary.Read(bufIO, binary.LittleEndian, DLT645Address)
    fmt.Println(DLT645Address)
```

```
    binary.Read(bufIO, binary.LittleEndian, DLT645Start)
    fmt.Println(DLT645Start)

    binary.Read(bufIO, binary.LittleEndian, ControlType)
    fmt.Println(ControlType)

    binary.Read(bufIO, binary.LittleEndian, DLT645DataLen)
    fmt.Println(DLT645DataLen)

    binary.Read(bufIO, binary.LittleEndian, DLT645DataCode)
    fmt.Println(DLT645DataCode)

    binary.Read(bufIO, binary.LittleEndian, DLT645Data)
    fmt.Println(DLT645Data)

    var number =""
    var DataLen = len(DLT645Data)
    for k = 0; k<DataLen;k++    {
        DLT645Data1[k] = DLT645Data[(DataLen-1)-k]
    }
    fmt.Println("DLT645Data1",DLT645Data1)
    for i = 0; i < len(DLT645Data1); i++ {
        var dd = toInt(DLT645Data1[i])-51
        fmt.Println(dd)
        number += fmt.Sprintf("%02X", dd)
    }

    fmt.Println(number)

    value, _ = strconv.Atoi(number)
    data = toFloat(toFloat(value) * 0.01)
    fmt.Println(data)
    SetDeviceData("测->645 总电能",data);
    binary.Read(bufIO, binary.LittleEndian, DLT645DataBlack)
    fmt.Println(DLT645DataBlack)


    time.Sleep(1000*time.Millisecond)
}
```

13.11 MQTT 客服端操作,订阅、发布消息

```
var MQTT = import("mqtt")

var time = import("time")
var fmt = import("fmt")

func subFunc(client , msg) {
	fmt.Println("主题",msg.Topic())
	fmt.Println("数据",msg.Payload())
}
var brokerHost = "mqtt://127.0.0.1:1883"

var opts = MQTT.NewClientOptions()
opts = opts.AddBroker(brokerHost)

opts.SetClientID("Hw-Link")
opts.SetUsername("Hw-Link")
opts.SetPassword("123456")
opts.SetKeepAlive(60 * 2 * time.Second)
opts.SetDefaultPublishHandler(subFunc)
var MqttClient = MQTT.NewClient(opts)
var token = MqttClient.Connect()

if token.Wait() && token.Error() != nil {
	fmt.Println("connect error")
}
var deviceOnlineOfflineTopic = "$SYS/brokers/+/clients/#"

var token1 = MqttClient.Subscribe(deviceOnlineOfflineTopic, 0, nil);

if    token1.Wait() && token1.Error() != nil {
	fmt.Printf("订阅设备的上下线消息失败  %s\n", deviceOnlineOfflineTopic)
}

for{
	//发布主题
	MqttClient.Publish("/sys/mqttx_9efe5836232/post", 0, false, "test")
	time.Sleep(1000*time.Millisecond)
}
```

13.12 Modbus Tcp Server

```
var modbus = import("modbus")
var fmt = import("fmt")
```

```
var binary = import("encoding/binary")

func ReadReadHoldingRegistersHander(reg,data){

    var address = binary.BigEndian.Uint16(data)
    fmt.Println("address",address)
    var quality = binary.BigEndian.Uint16(data[2:])
    fmt.Println("quality",quality)
    var setData = make([]byte, 2)
    setData[0] = 12
    setData[1] = 13
    var er = reg.WriteHoldingsBytes(address, 1, setData)
    var value, err = reg.ReadHoldingsBytes(address, quality)
    a = make([]byte, len(value)+1)
    a[0] = len(value)
    var j = 1
    for i = 0; i < len(value); i++ {
       a[j] = value[i]
       j++
    }
    return a,err
}

func ReadInputRegistersHander(reg,data){

    var address = binary.BigEndian.Uint16(data)
    fmt.Println("address",address)
    var quality = binary.BigEndian.Uint16(data[2:])
    fmt.Println("quality",quality)
    var setData = make([]byte, 2)
    setData[0] = 12
    setData[1] = 13
    var er = reg.WriteHoldingsBytes(address, 1, setData)
    var value, err = reg.ReadHoldingsBytes(address, quality)
    a = make([]byte, len(value)+1)
    a[0] = len(value)
    var j = 1
    for i = 0; i < len(value); i++ {
       a[j] = value[i]
       j++
    }
    return a,err
}
```

```
func WriteSingleRegisterHander(reg,data){


    return a,err
}

func ReadCoilsHander(reg,data){


    return a,err
}
func ReadDiscreteInputsHander(reg,data){


    return a,err
}

func WriteSingleCoilHander(reg,data){


    return a,err
}

var srv = modbus.NewTCPServer()
srv.AddNodes(modbus.NewNodeRegister(1,0, 10, 0, 10,0, 10, 0, 10))
srv.RegisterFunctionHandler(1,ReadCoilsHander)
srv.RegisterFunctionHandler(2,ReadDiscreteInputsHander)

srv.RegisterFunctionHandler(5,WriteSingleCoilHander)

srv.RegisterFunctionHandler(4,ReadReadHoldingRegistersHander)
srv.RegisterFunctionHandler(3,ReadInputRegistersHander)

srv.RegisterFunctionHandler(6,WriteSingleRegisterHander)


var err11 = srv.ListenAndServe(":502")
if err11 != nil {
    fmt.Println(err11)
}
```

13.13 Websocket 服务

```
var http = import("net/http")
```

```
var websocket = import("websocket")
var fmt = import("fmt")

func CheckOrigin (r ) {
        return true
}
var upGrader = new(websocket.Upgrader)
fmt.Println("upGrader:",upGrader)
upGrader.CheckOrigin = CheckOrigin

http.HandleFunc("/", func(w, r) {

   var conn,err = upGrader.Upgrade(w,r,nil)

   go func(){
      for{
         mtype,msg,err = conn.ReadMessage()
         conn.WriteMessage(mtype,msg)
      }
   }(conn)

})

http.ListenAndServe(":8099", nil)
```

13.14 提供 API 服务，测试实时数据和数据库数据

```
var http = import("net/http")
var json = import("encoding/json")
var fmt = import("fmt")

http.HandleFunc("/DiyChartData", func(w, r) {
      w.Header().Set("Content-Type", "application/json")
      w.Header().Set("Access-Control-Allow-Origin", "*")
      w.Header().Set("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS")
      w.Header().Set("Access-Control-Allow-Headers", "content-type")
      w.Header().Set("Access-Control-Expose-Headers", "Access-Control-Allow-Headers, Token")
      w.Header().Set("Access-Control-Allow-Credentials", "true")
      responseStu = make(struct {
         A int64,
         B float64
      })
      responseStu.A = GetDeviceData("ww->register0")
      var responseData,_ = json.Marshal(responseStu)
```

```
	w.Write(responseData)
})

http.HandleFunc("/DiyChartDataFromDb", func(w, r) {
	w.Header().Set("Content-Type", "application/json")
	w.Header().Set("Access-Control-Allow-Origin", "*")
	w.Header().Set("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS")
	w.Header().Set("Access-Control-Allow-Headers", "content-type")
	w.Header().Set("Access-Control-Expose-Headers", "Access-Control-Allow-Headers, Token")
	w.Header().Set("Access-Control-Allow-Credentials", "true")
	y                                                                                    =
GetRemoteDbData("mysql","ismctl","h1ofTL636j","sql.s1267.vhostgo.com",3306,"ismctl","SELEC
T * FROM devices_alarm_list")
	var responseData,_ = json.Marshal(y)
	w.Write(responseData)
})

http.ListenAndServe(":8099", nil)
```

13.15 API 服务，获取请求的参数

```
var time = import("time")
var fmt = import("fmt")
var http = import("net/http")
var json = import("encoding/json")
var io =    import("io")
var ioutil =    import("io/ioutil")

//跨域开启
func openky(w){
	w.Header().Set("Content-Type", "application/json")
	w.Header().Set("Access-Control-Allow-Origin", "*") //允许访问所有域
	w.Header().Set("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS")
	w.Header().Set("Access-Control-Allow-Headers", "content-type")
	w.Header().Set("Access-Control-Expose-Headers", "Access-Control-Allow-Headers, Token")
	w.Header().Set("Access-Control-Allow-Credentials", "true")
}

//测试请求数组获取
http.HandleFunc("/getdemo", func(w, r) {
	openky(w);
	//读取请求体信息
```

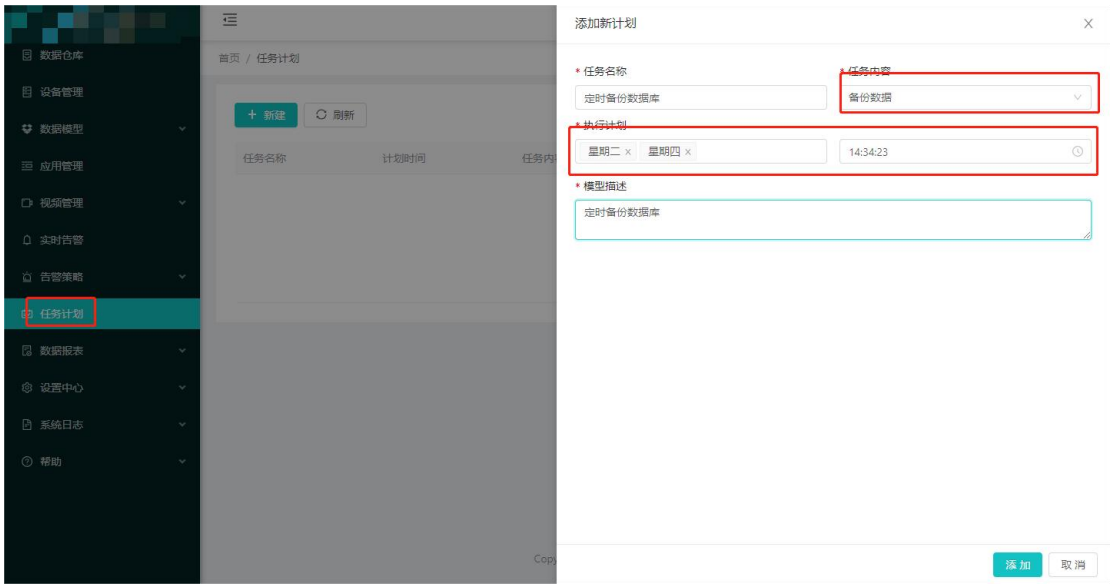
```
    rrs = getObjSZ();

     //实现 json 对象向 go 结构体对象转换，把请求的参数转为 json 对象
    bodyContent, err = ioutil.ReadAll(r.Body)
    err1 = json.Unmarshal(bodyContent, &rrs)

    //
    if    err1 != nil {
          fmt.Printf("Unmarshal err, %v\n", err1)
          return
     }

    fmt.Println(rrs)
    w.Write(toByteSlice("OK"))
})

//创建一个结构体数组
func getObjSZ(){

    rr = make( struct {
          Info      map[string]string ,//map 对象保存 json 信息
          Shuzhu    []map[string]string//map 数组处理 json 数组
         })

    return rr;
}

http.ListenAndServe(":8091", nil)
```

十四、任务计划

任务计划是 Hw-Link 定时型的周期运行的任务，可以帮助用户备份数据库，定时设置设备的数据，删除不需要的历史数据等

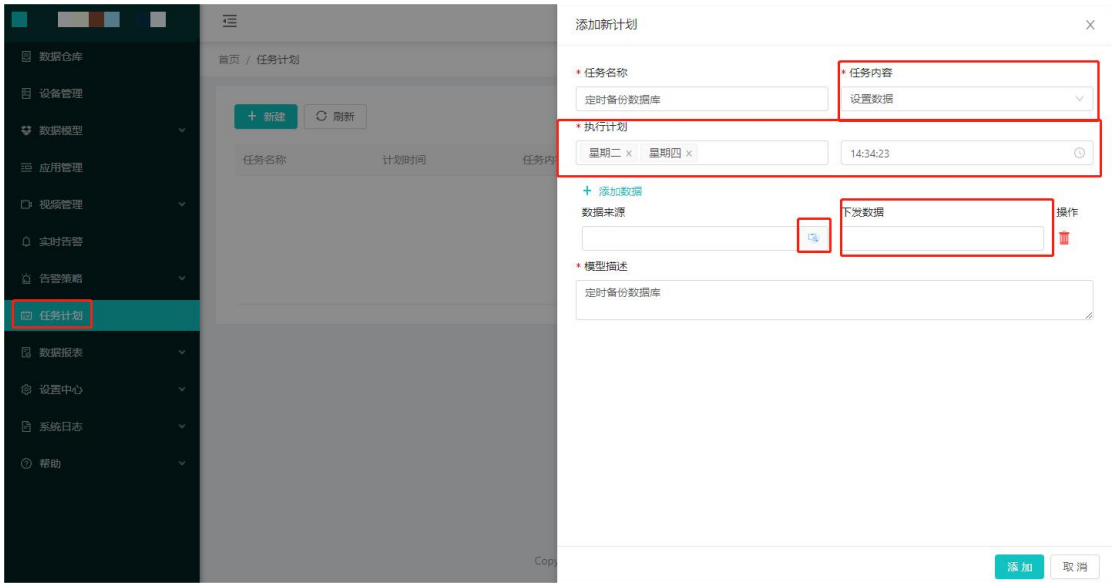
14.1 备份数据库

任务内容选择备份数据，选择好执行的时间，Hw-Link 就会按照此计划进行备份数据库的操作。



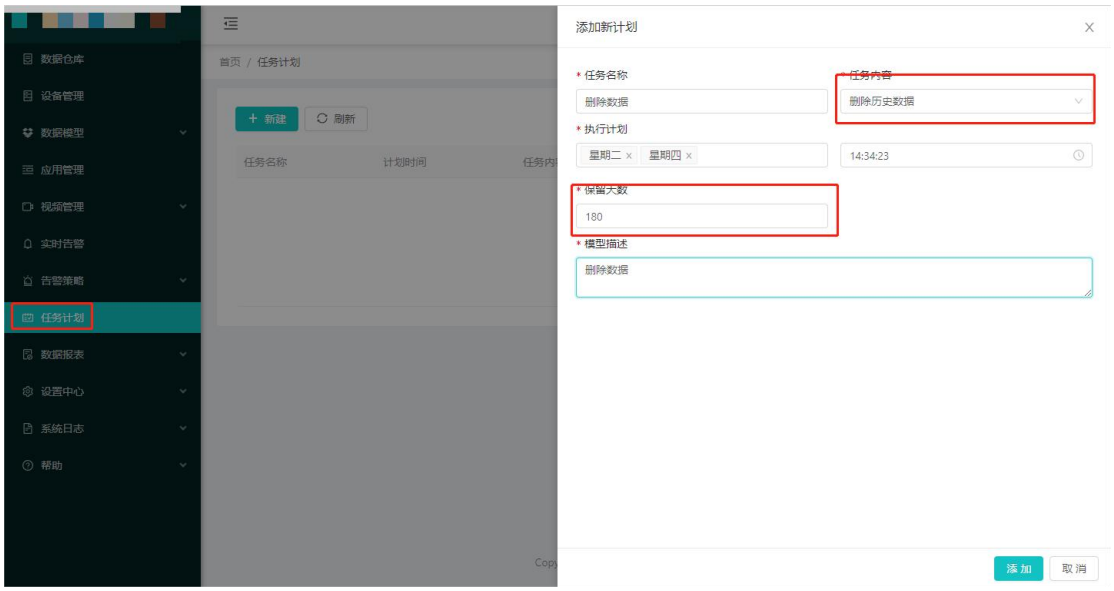
14.2 设置数据

如果想让 Hw-Link 定时去设置一些设备的数据，任务内容选择设置数据，然后选择要设置的设备的数据和值，Hw-Link 会根据时间自动设置填写的值到设备中去



14.3 删除数据

分为删除历史数据和历史告警数据，任务内容选择删除历史数据，保留天数 180，Hw-Link 就自动删除 180 天之前的数据。

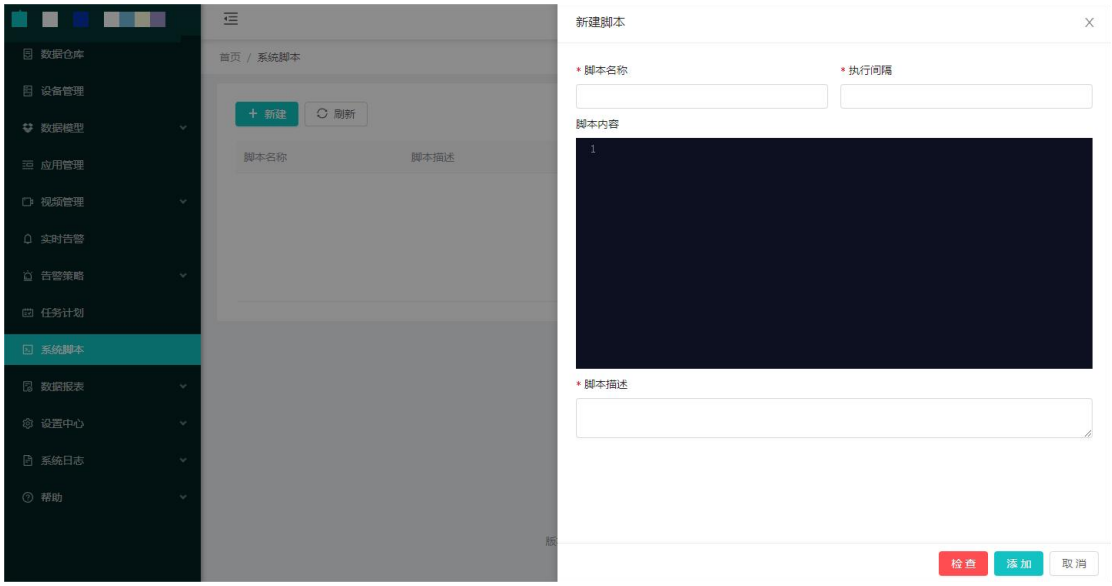


十五、系统脚本

为了解决多设备数据运算和多设备数据联动，Hw-Link 提供了类 go 语言的格式的系统脚本

15.1 新建脚本

执行间隔是脚本的执行间隔时间



输入完脚本后，点击检查，可以检查脚本是否有错误。

15.2 脚本函数

GetDeviceData("设备名->数据名") 获取设备的数据

相关信息

参数格式：

设备名->数据名

SetDeviceData("数据名",数据) 设置设备的数据

相关信息

参数格式

参数 1：设备名->数据名，

参数 2：浮点数

GetModuleDevice("模型名称") 获取系统所有绑定的数据模型设备

相关信息

参数:

模型名称

结果:

返回的数据是对象数组

例子

```
g = GetModuleDevice("tcp")

for i in g {
   Print(i.Name)
   f = i.Name+"->功率 1"
   f1 = i.Name+"->功率 2"
   f2 = i.Name+"->功率和"
   y = GetDeviceData(f)
   x = GetDeviceData(f1)
   SetDeviceData(f2,x+y)
}
```

RecordVideo("",100,"path")函数

相关信息

第一个参数：视频名称(这个是添加的视频名称)

第二个参数：录像的时间，单位为秒

第三个参数：保存路径 注意 windows 下面的路径有反斜杠，要这样表达两个反斜杠表达一个反斜杠

Print（变量名） 打印函数

GetRemoteDbData("数据库类型","用户名","密码"，“主机”，端口，“数据库名称”，“SQL 语句”)

相关信息

参数：

数据库类型：目前只支持 mysql

用户名:连接数据库的用户

密码:连接数据库的用户的密码

主机:数据库主机

端口:数据库主机端口

数据库名称:连接的数据库

SQL 语句:执行的 SQL 语句

例子：

```
y = GetRemoteDbData("mysql","test","pwd123456","127.0.0.1",3308,"ism","select id ,count from alarm_notice where uuid='feae-dead-eead-erfb'")

for i in y {
   Print(i.Name)
   Print(y.id)
   Print(y.count)
}
```

BitGet("t->BitValue",5)函数 获取一个整型数据的每个位的值。

相关信息

第一个参数：设备里的数据，格式:设备名->数据名，如 t->BitValue

第二个参数：要求获取数据的第几位，如第 6 位，BitGet("t->BitValue",6)

BitSet("t->BitValue",5,1)函数 设置一个整型数据的某个位的值。

相关信息

第一个参数：设备里的数据，格式:设备名->数据名，如 t->BitValue

第二个参数：要求设置数据的第几位

第三个参数：设置此位的值，取值 0 和 1

15.3 语法

15.3.1 定义变量

```
x = 1
y = x + 1
```

15.3.2 条件

```
if x < 1 || y < 1 {
        println(x)
} else if x < 1 && y < 1 {
        Print(y)
} else {
        Print(x + y)
}
```

15.3.3 切片

```
a = []interface{1, 2, 3}
Print(a) // [1 2 3]
Print(a[0]) // 1
```

15.3.4 键值对

```
a = map[interface]interface{"x": 1}
Print(a) // map[x:1]
a.b = 2
a["c"] = 3
Print(a["b"]) // 2
Print(a.c) // 3
```

15.3.5 结构体

```
a = make(struct {
        A int64,
        B float64
})
a.A = 4
a.B = 5.5
Print(a.A) // 4
Print(a.B) // 5.5
```

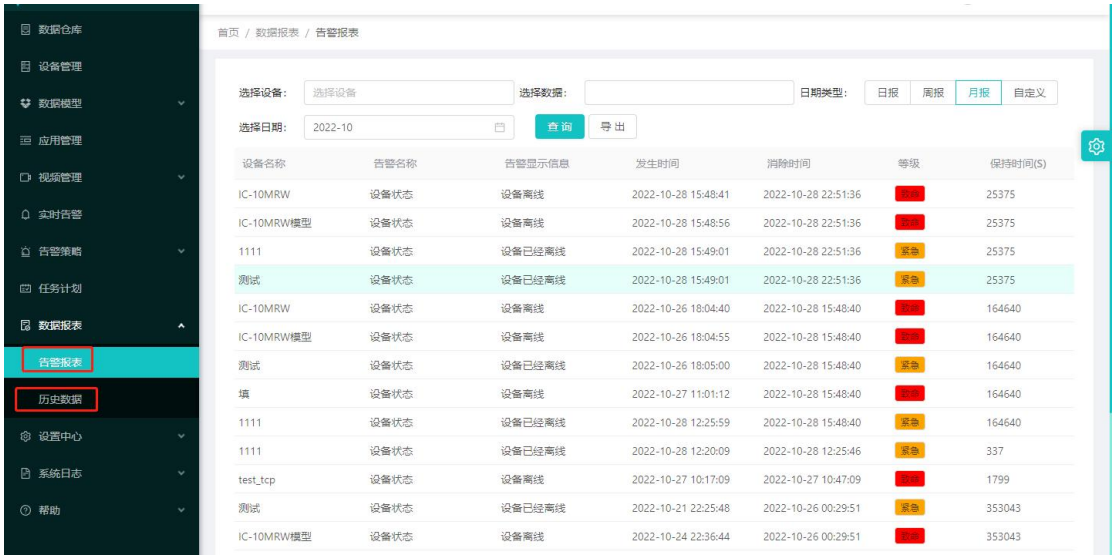
15.3.6 函数

```
func a (x) {
        Print(x + 1)
}
a(5) // 6
```

十六、数据报表

数据报表是 Hw-Link 根据用户的配置自动存储的数据，用户可以根据时间和设备数据进行查询和导出

导出按钮可以在线把数据导出到表格中，并下载。

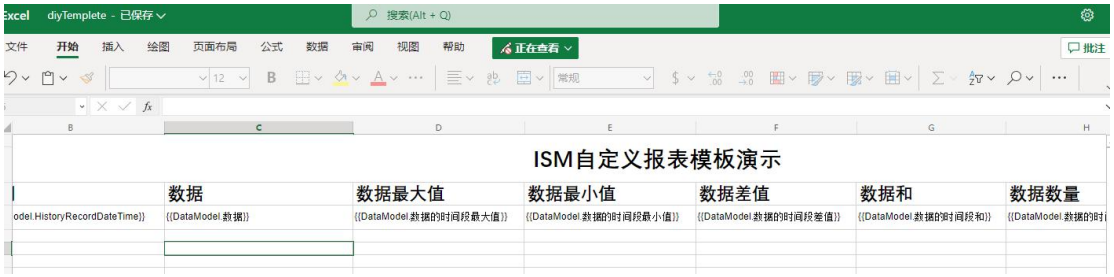


十七、自定义报表

17.1 自定义报表的使用

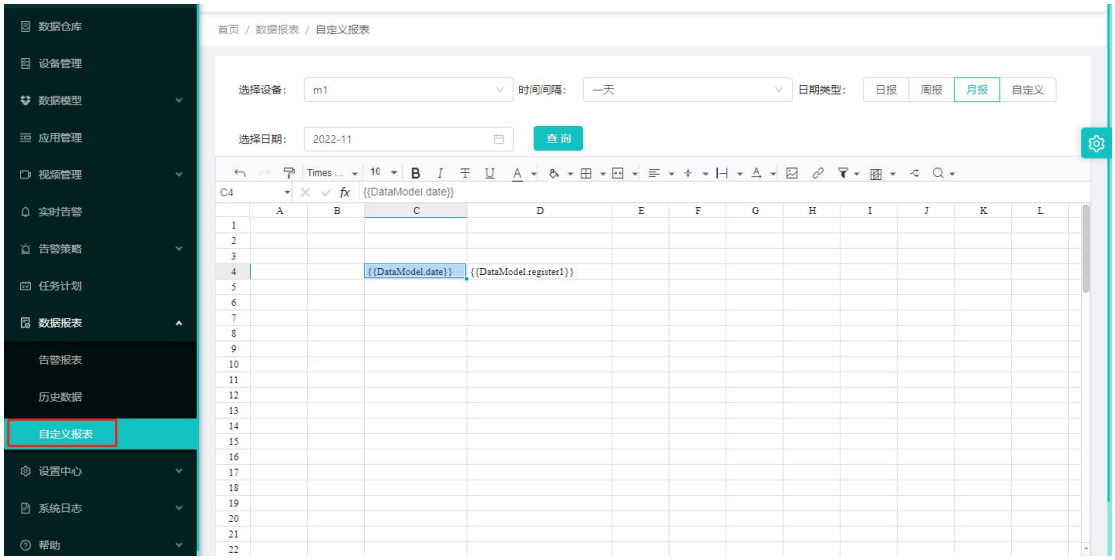
Hw-Link 提供简易化的在线自定义报表数据的导出功能。

自定义报表演示模板下载



17.2 打开自定义报表

登录 Hw-Link，找到自定义报表



右击单元格，有四个变量可以使用，

数据变量: 历史数据中的数据名称，也就是要导出的数据

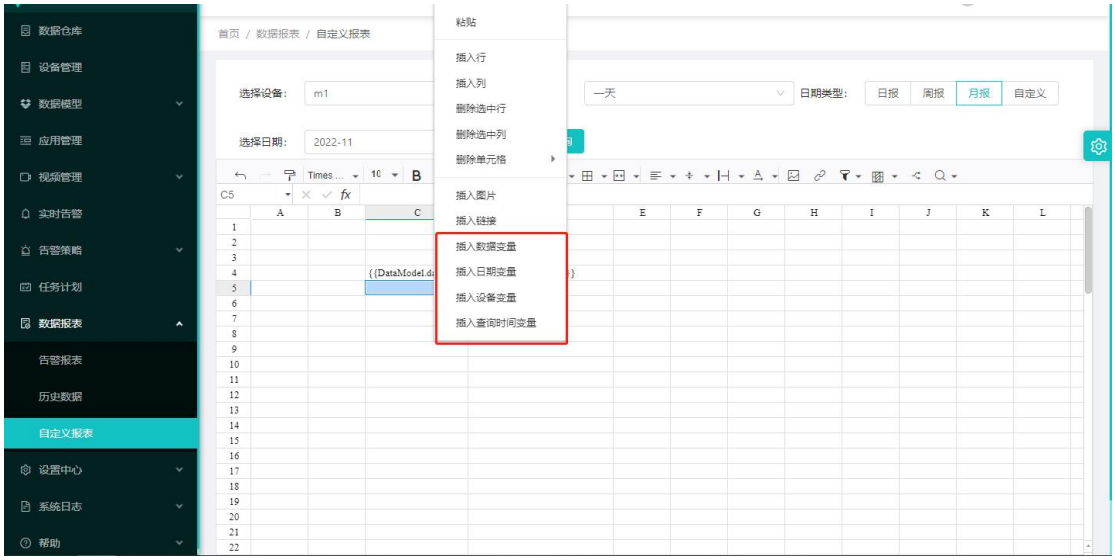
日期变量: 历史数据中的数据数据记录时间

设备变量：是查询的设备名称

查询时间：是查询条件中的时间范围

系统变量:

数据变量加上"的时间段最大值"、"的时间段最小值"、"的时间段差值"、"的时间段和"、"的时间段数量"、"的时间段平均值"。分别可以导出时间段内的系统计算出的数据



排版后即可点击查询按钮，导出你想要的模板

十八、设置中心

18.1 告警通知

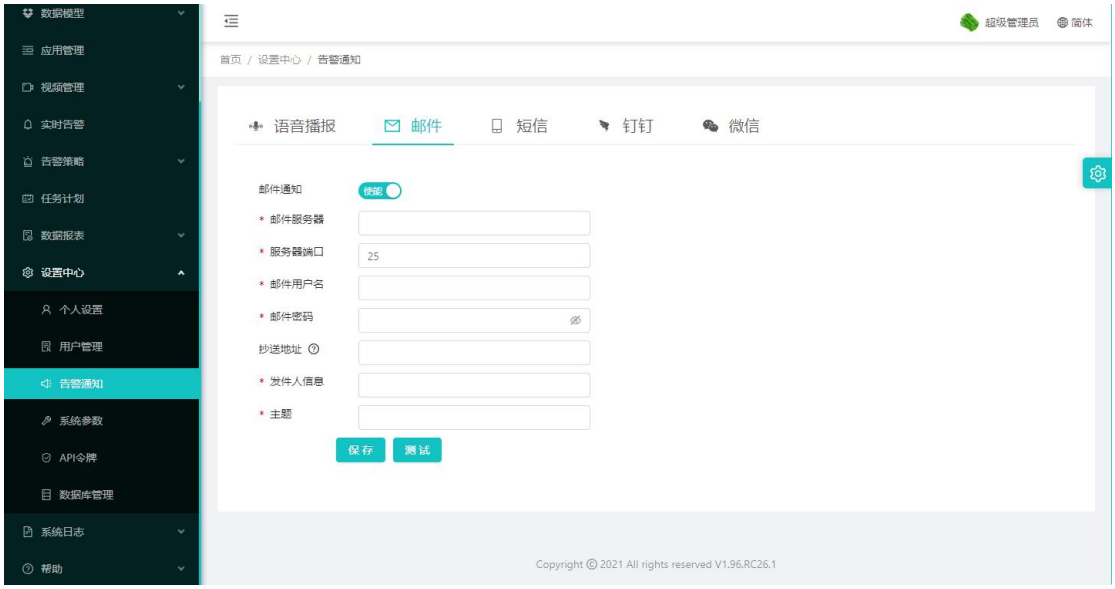
Hw-Link 有多种告警通知方式，分别是语言播报、邮件、短信、钉钉、微信，下面讲一一介绍每个通知方式。

18.1.1 语言播报

Hw-Link 告警时会主动语音播报告警的内容，这个很好理解。

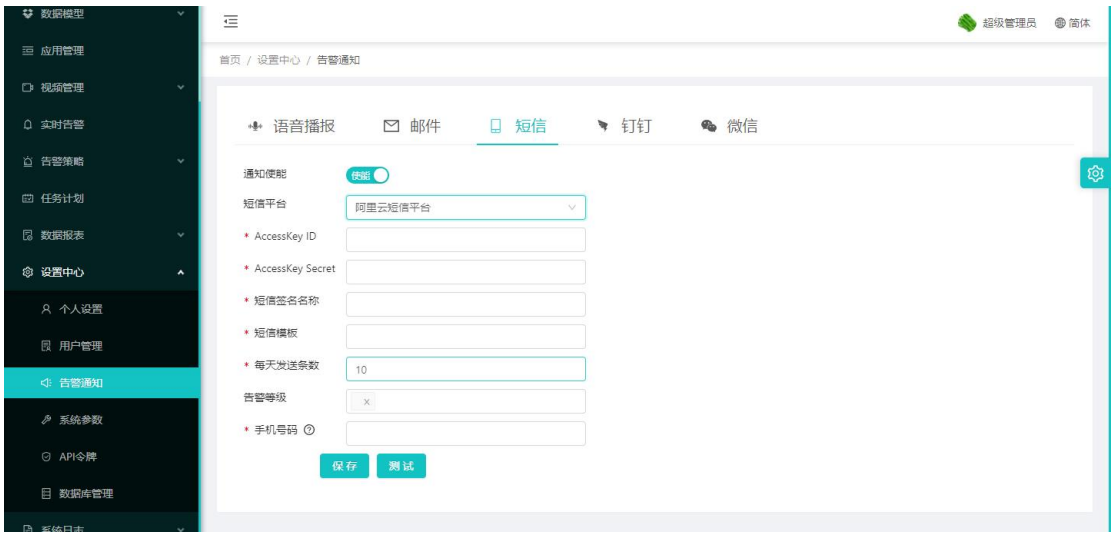
18.1.2 邮件通知

设置好邮件服务器，发件人的信息，抄送的地址，Hw-Link 就会根据设置告警时自动发邮件到指定的抄送地址。

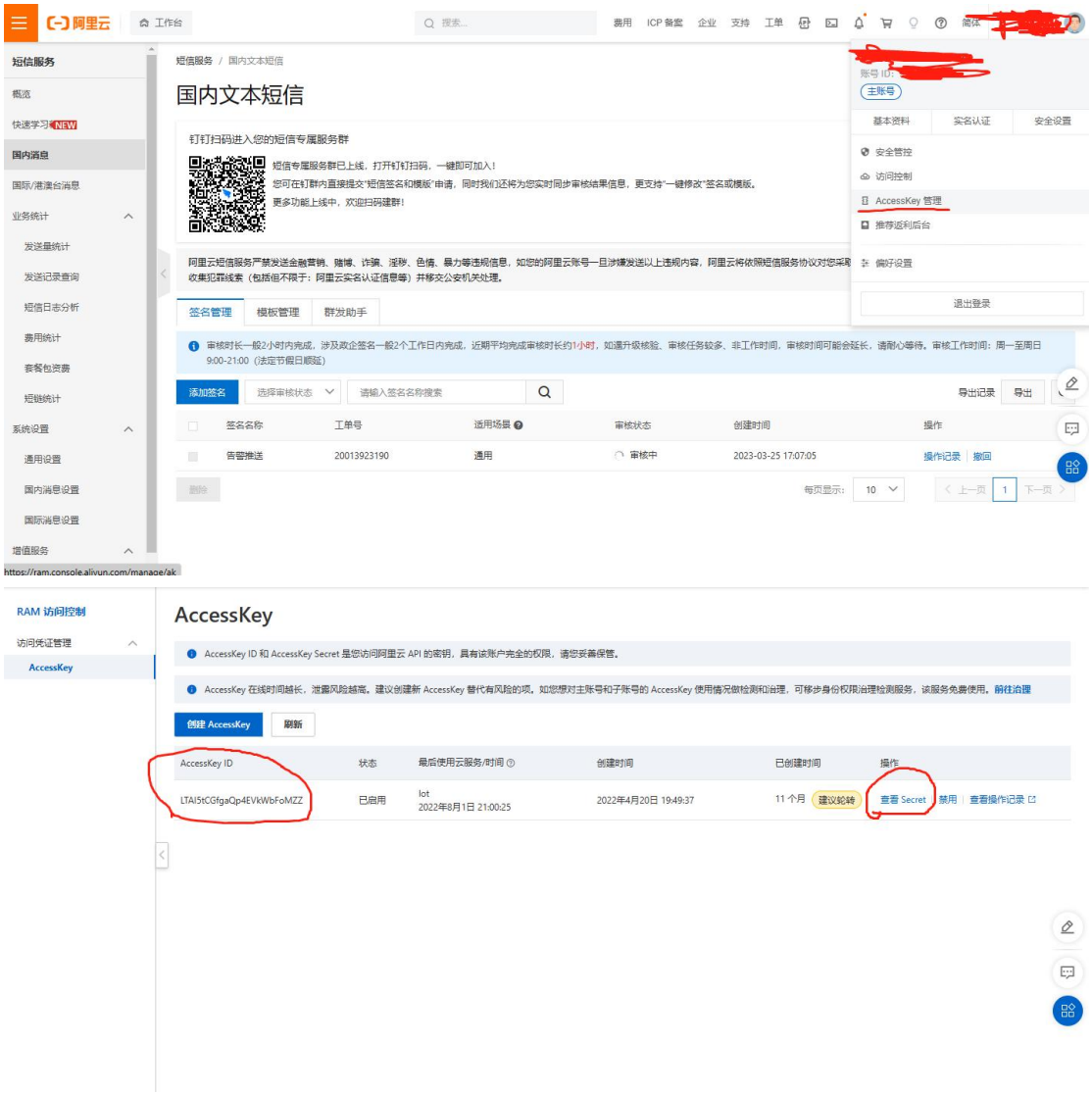


18.1.3 短信

目前只支持阿里云平台，用户自己到阿里云申请短信服务，按照页面的提示配置好短信服务，Hw-Link 就会主动告警内容到指定的手机号码，多个号码以;分割



18.1.4 AccessKey ID & AccessKey Secret



18.1.5 短信签名

18.1.6 短信模板

18.1.7 短信模板变量

相关信息

短信模板选择通知短信

模板变量中添加变量，如下:

${device_name}:设备名称

${data_name}:数据名称

${alarm_message}:告警消息

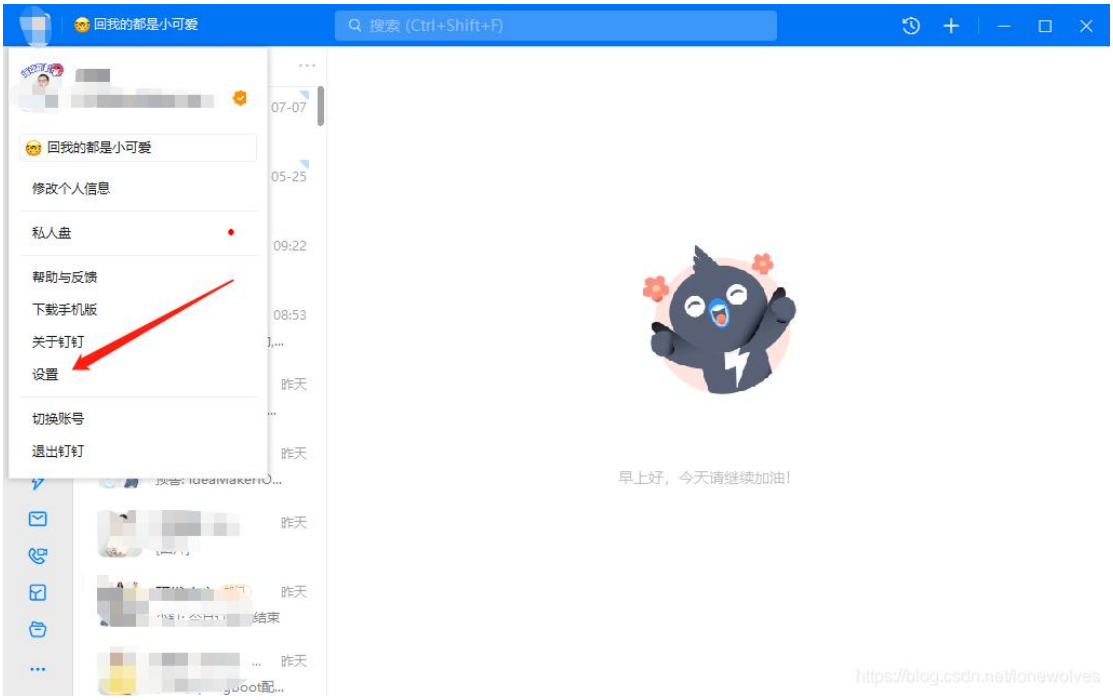
${happen_time}:发生时间

其他根据自己的情况填写。

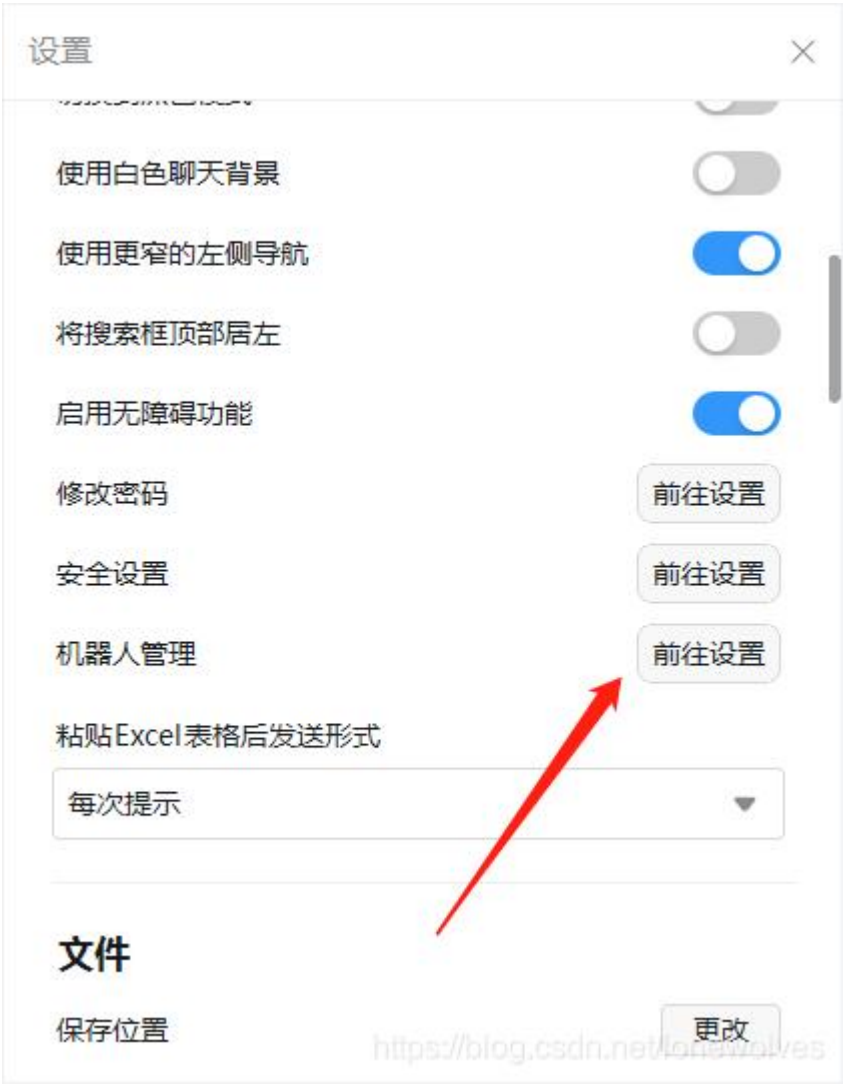
18.1.8 钉钉

先创建一个自定义钉钉机器人。

在头像里面点击设置



然后找到机器人管理，前往设置



打开后自定义一个机器人并设置好安全设置

把 Webhook 地址和加签的秘钥填入界面中，Hw-Link 支持多个钉钉群的通知。

18.1.9 微信

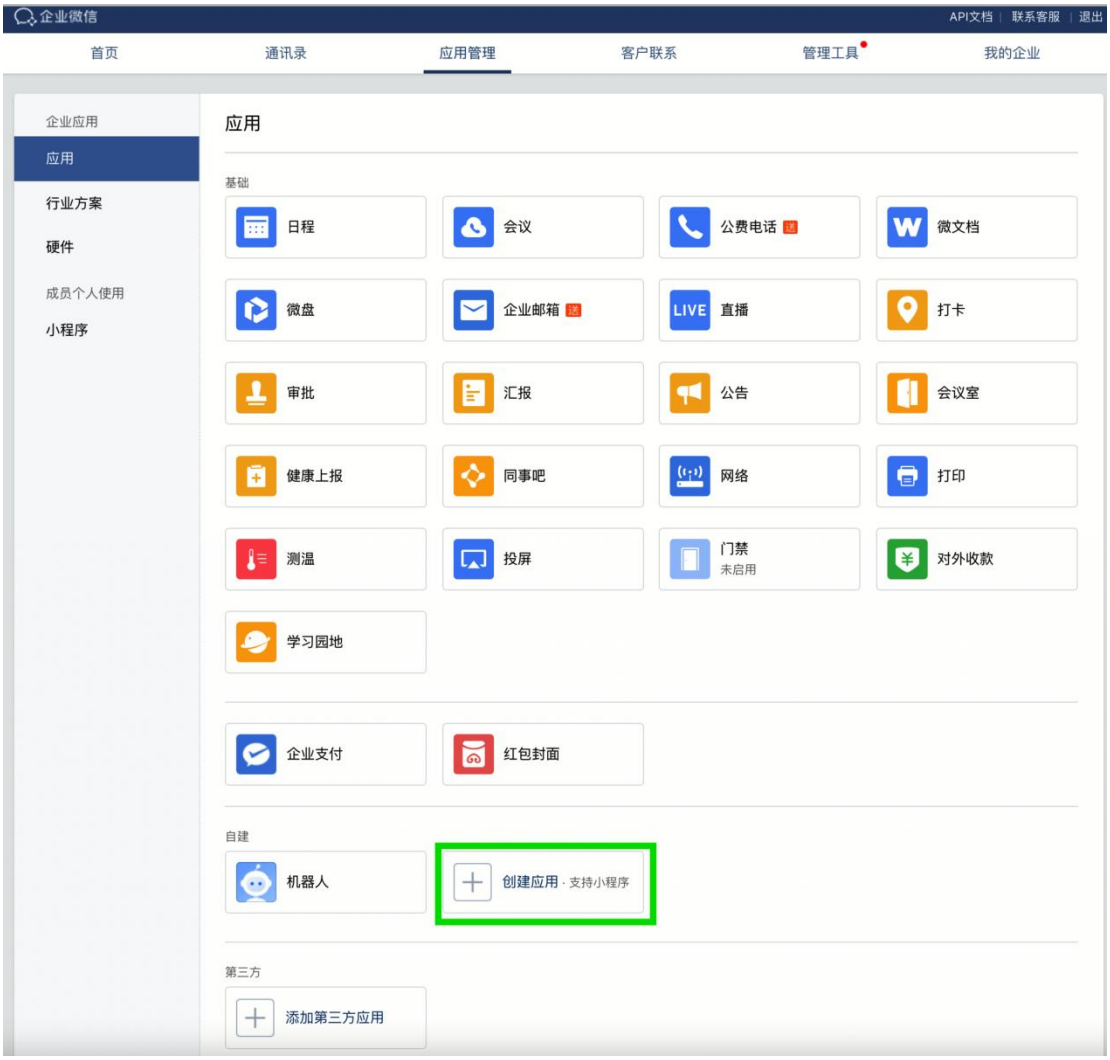
18.1.9.1 准备工作

您需要准备一个企业微信帐号。

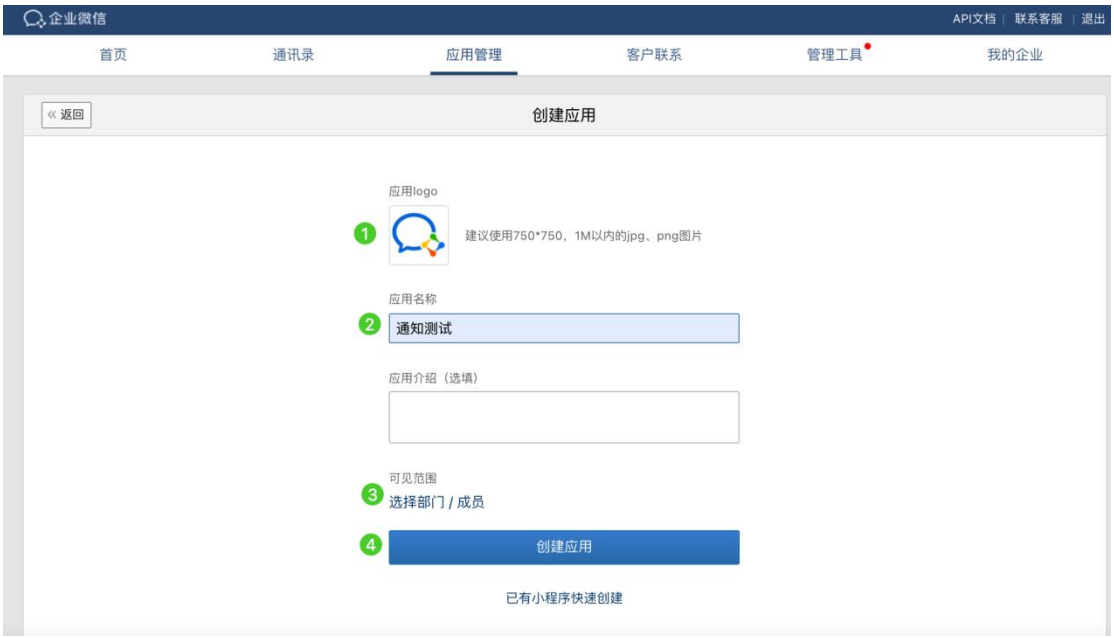
18.1.9.2 动手实验

步骤 1：创建应用

1. 登录企业微信管理后台，点击**应用管理**。
2. 在**应用管理**页面，点击**自建**下的**创建应用**。



3. 在**创建应用**页面，上传应用 Logo、输入应用名称（例如，通知测试），点击**选择部门 / 成员**编辑**可见范围**，然后点击**创建应用**。



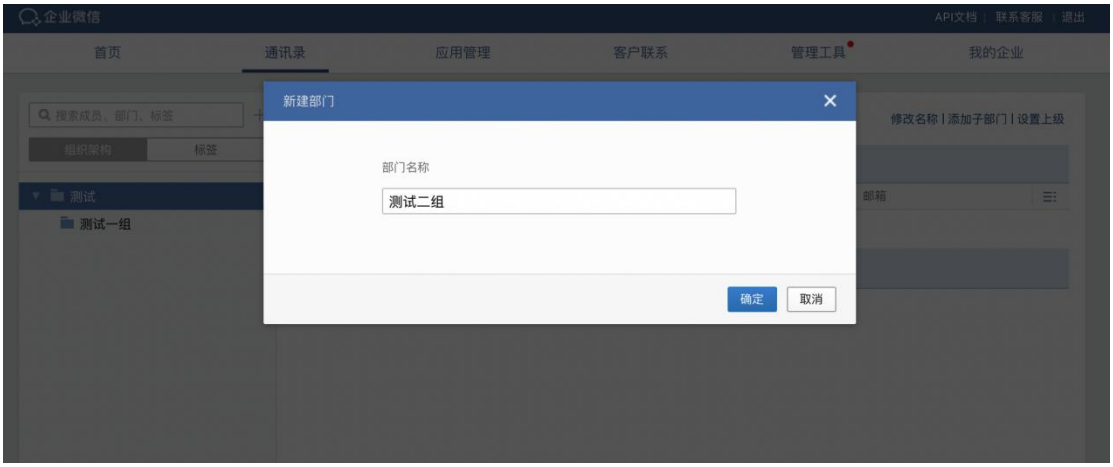
备注请确保将需要接收通知的用户、部门或标签加入可见范围中。

4. 应用创建完成后即可查看其详情页面，**AgentId** 右侧显示该应用的 ID。点击 **Secret** 右侧的**查看**，然后在弹出对话框中点击**发送**，便可以在企业微信客户端查看 Secret。此外，您还可以点击**编辑**来编辑可见范围。

步骤 2：创建部门或标签

1. 在**通讯录**页面的**组织架构**选项卡下，点击**测试**（本教程使用测试部门作为示例）右侧的 

2. 在弹出对话框中，输入部门名称（例如测试二组），然后点击**确定**。

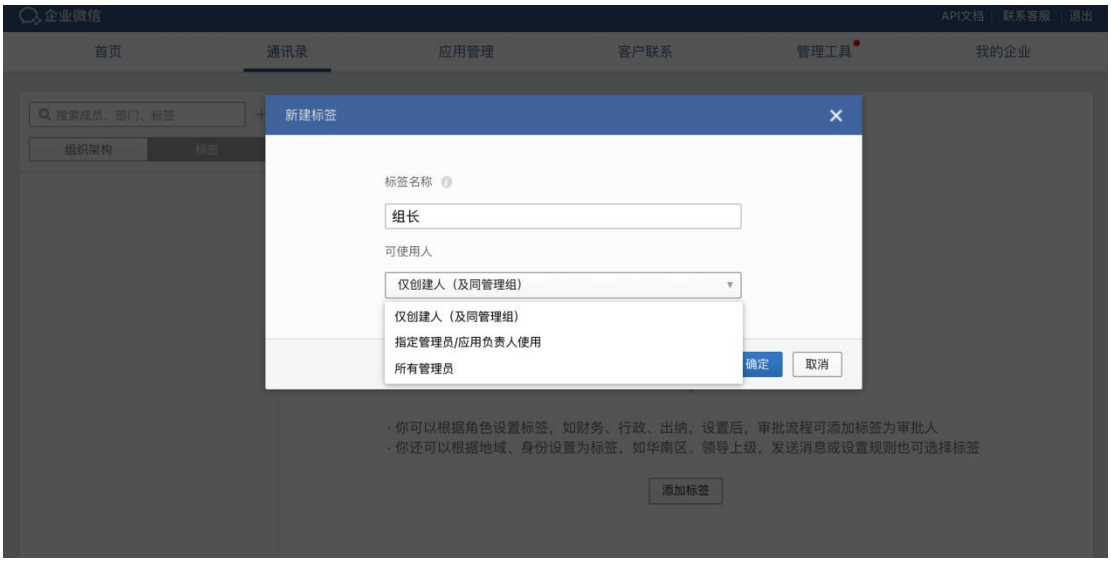


3. 创建部门后，您可以点击右侧的**添加成员**、**批量导入**或**从其他部门移入**来添加成员。添加成员后，点击该成员进入详情页面，查看其帐号。

4. 您可以点击测试二组右侧的 

5. 点击**标签**选项卡，然后点击**添加标签**来创建标签。若管理界面无**标签**选项卡，请点击加号图标来创建标签。

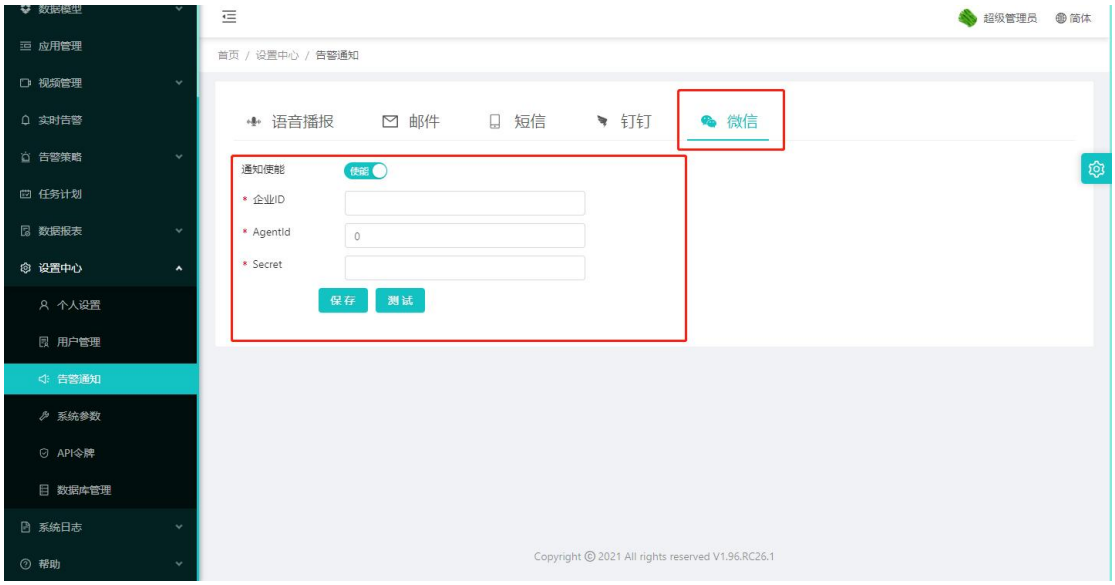
6. 在弹出对话框中，输入标签名称，例如组长。您可以按需指定**可使用人**，点击**确定**完成操作。



7. 创建标签后，您可以点击右侧的**添加部门/成员**或**批量导入**来添加部门或成员。点击**标签详情**进入详情页面，可以查看此标签的 ID。

8. 要查看企业 ID，请点击**我的企业**，在**企业信息**页面查看 ID。

把信息中的企业 ID，AgentId,Secret 填入 Hw-Link 中，Hw-Link 就会自动推送告警信息到微信中。



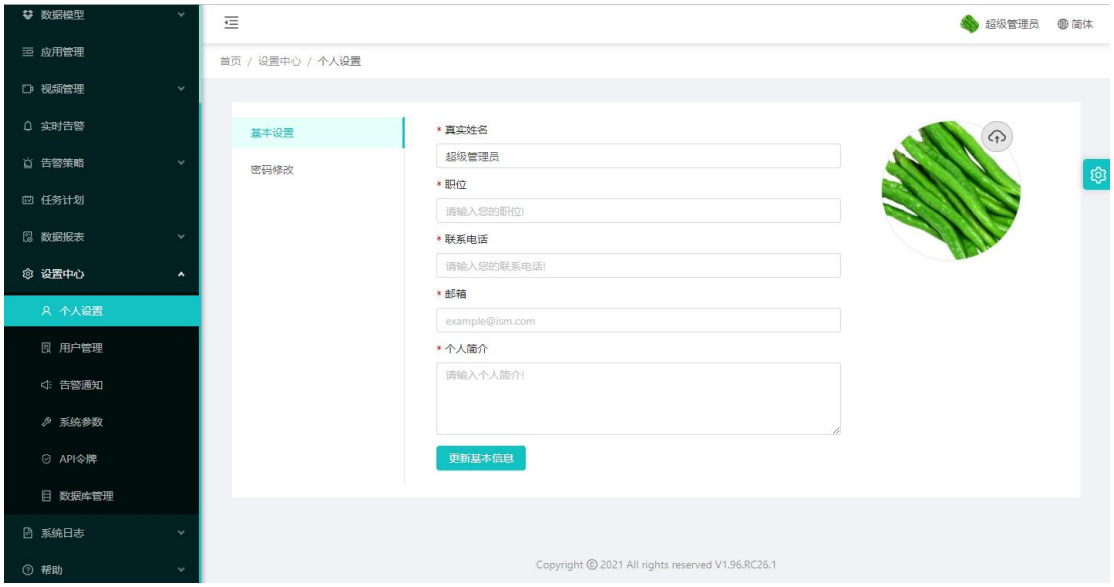
企业的可信认证

登录企业微信后台，选择「应用管理」，选择对应的应用，设置「企业可信 IP」。

如果客户遇到添加 IP，提示是第三方服务商 IP（实际不是），需要找企业微信客服验证一下企业可信 IP，如下图所示：

18.2 个人设置

个人设置就是设置当前登录用户的信息，用户可以根据界面自行设置



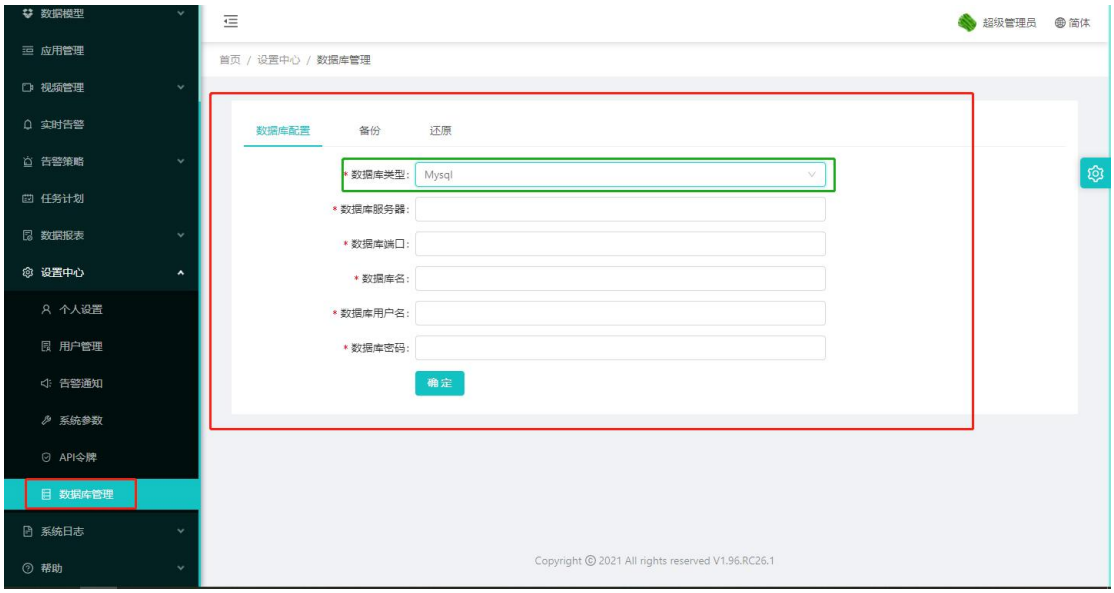
18.3 数据库管理

Hw-Link 默认使用的 Sqlite3 数据库，SQLite，是一款轻型的数据库，是遵守 ACID 的关系型数据库管理系统，它包含在一个相对小的 C 库中。它是 D.RichardHipp 建立的公有领域项目。它的设计目标是嵌入式的，而且已经在很多嵌入式产品中使用了它，它占用资源非常的低，在嵌入式设备中，可能只需要几百 K 的内存就够了。它能够支持 Windows/Linux/Unix 等等主流的操作系统，同时能够跟很多程序语言相结合，比如 Tcl、C#、PHP、Java 等，还有 ODBC 接口，同样比起 Mysql、PostgreSQL 这两款开源的世界著名数据库管理系统来讲，它的处理速度比他们都快。

Hw-Link 支持 MySQL 数据库，可以自由切换，只需要在界面上选择即可。

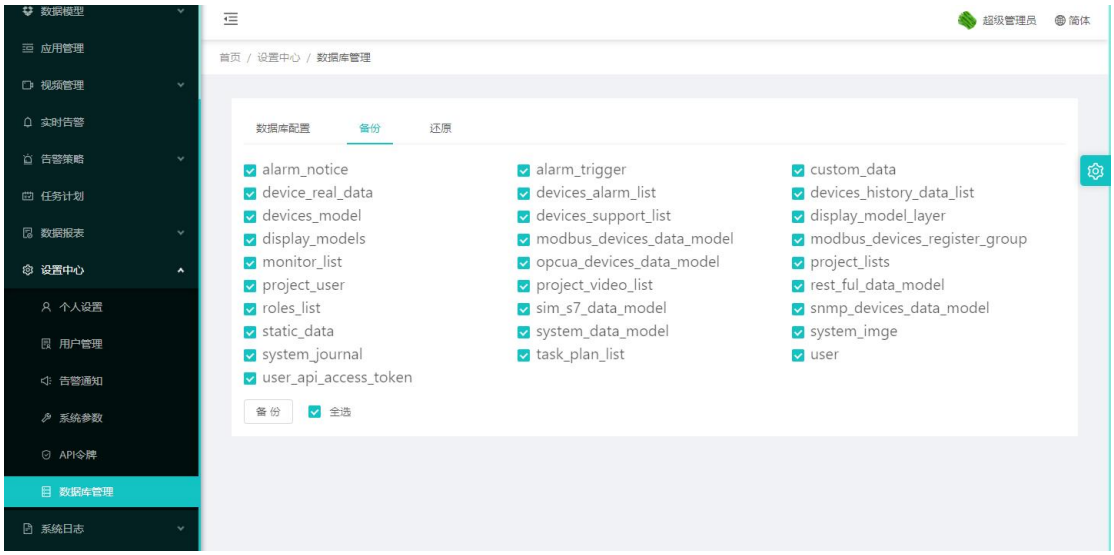
18.3.1 切换 MySQL 数据库

在界面上选择 MySQL，根据界面上提示的内容，填写 MySQL 的数据库信息。



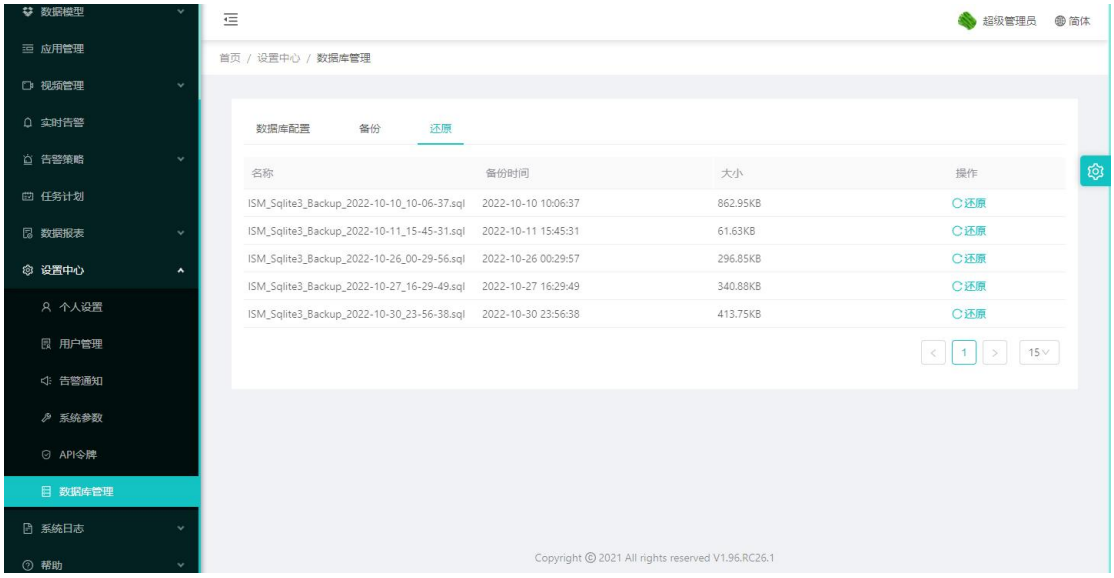
18.3.2 备份数据库

用户可以手动选择备份数据，点击备份按钮，Hw-Link 会在自动按照数据库表备份数据。



18.3.3 还原数据库

用户可以根据需要手动还原已经备份的数据库数据，找到数据库文件，然后点击还原即可。



十九、用户管理

Hw-Link 系统支持多个用户同时登录，系统管理员可以添加用户，其他用户不能添加。可以根据需要自行添加

可以添加的角色有两种，一个是普通用户，一个是操作员。

普通不能登录后台管理界面，只能浏览发布的应用

操作员跟管理员的权限差不多

二十、开发实时数据库图标组件

20.1 开发 API 接口

功能说明

此文章主要讲述通过 Hw-Link 的 Go 虚拟机提供接口，接口提供了获取实时数据的返回格式，给前端图表使用。这篇文章需要相关 Go 语言和 API 接口相关的知识，不了解的建议先看看网上的相关说明。

20.1.1 第一步 构建 APi 接口

我们首先把我们要展示的数据通过 API 接口开放出去，这样就可以通过接口把采集的实时数据开放出去给其他需要的组件调用。

在安装目录下面的 sys_script 目录下面新建 api.go 文件，把下面的代码赋值到文件中，然后重启 Hw-Link.

下面的代码只是演示，具体的功能要根据自己的业务需求修改

```
var time = import("time")var fmt = import("fmt")var http = import("net/http")var json = import("encoding/json")var io =  import("io")var ioutil =  import("io/ioutil")
//跨域开启，此段不需要修改 func openky(w){
    w.Header().Set("Content-Type", "application/json")
    w.Header().Set("Access-Control-Allow-Origin", "*") //允许访问所有域
    w.Header().Set("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS")
    w.Header().Set("Access-Control-Allow-Headers", "content-type")
    w.Header().Set("Access-Control-Expose-Headers", "Access-Control-Allow-Headers, Token")
    w.Header().Set("Access-Control-Allow-Credentials", "true")}
//请求接口 ApiDemo 是接口名称,可以修改为你们自己的接口名称
http.HandleFunc("/ApiDemo", func(w, r) {

    //此处加上跨域请求，每个接口都要先调用这个函数。
    openky(w);

    //构建返回的接口，格式为 json 格式，Hw-Link 的 Go 虚拟机语法，定义变量可以直接用变量赋值
    //不需要 var 声明。不能使用 := 去赋值
    //切记结构体的成员变量首字母一定要大写。

  //获取到值后的回复变量
  ResponseData = map[string]interface{}
  //失败后的消息返回变量
  FailedResponseData = map[string]interface{}

  //通过 GetDeviceData 函数把系统采集到数据赋值给结构体变量
  //m1 是测试的设备名称，register0 和 register1 是 m1 设备下面的两个数据名称
```

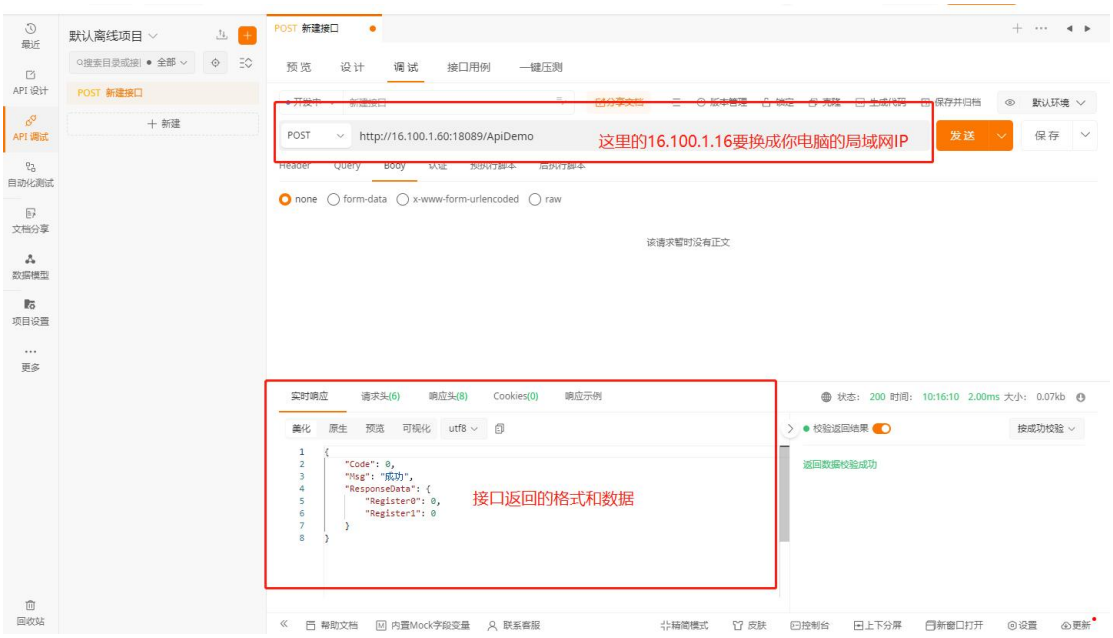
```
    // ResponseData["Register0"] = GetDeviceData("m1->register0")
    // ResponseData["Register1"] = GetDeviceData("m1->register1")
    //可以自己按照接口要求返回特定格式，此段代码返回的格式
    // {
    //     "Code": -2,
    //     "Msg": "设备值获取错误",
    //     "ResponseData": {
    //         "Register0": 0,
    //         "Register1": 0
    //     }
    // }
    ResponseData["ResponseData"] = map[string]interface{}
    ResponseData["ResponseData"]["Register0"]= GetDeviceData("m1->register0")
    ResponseData["ResponseData"]["Register1"]= GetDeviceData("m1->register1")
    deviceStatus = GetDeviceData("m1->设备状态")

    //GetDeviceData 函数如果没有获取到值，返回 null 空值

    if ResponseData["ResponseData"]["Register0"]==nil || ResponseData["ResponseData"]["Register1"]==nil{
        ResponseData["Code"]=-2
        ResponseData["Msg"]="设备值获取错误"
    }else if deviceStatus==0 {
        ResponseData["Code"]=-3
        ResponseData["Msg"]="设备离线"
    }else{
        ResponseData["Code"] = 0
        ResponseData["Msg"]="成功"
    }

    //把数据转化为 json 格式 json.Marshal 函数是把一个任意变量转为 json 格式
    responseJsonData,err = json.Marshal(ResponseData)
    if err!= nil{
        FailedResponseData["Code"]=-1
        FailedResponseData["Msg"]="JSON 格式转换失败"
        responseJsonData,_ = json.Marshal(FailedResponseData)
        //通过 w.Write 函数把数据通过 http 协议的 GET 或者 POST 方式返回
        w.Write(toString(responseJsonData))
    }else{
        //通过 w.Write 函数把数据通过 http 协议的 GET 或者 POST 方式返回
        w.Write(toString(responseJsonData))
    }
    })//18091 监听的端口，可以根据需要更改
fmt.Println("API 接口服务已经启动,提供了 ApiDemo")
```

```
http.ListenAndServe(":18089", nil)
```

20.1.2 第二步 使用 APiPost 测试接口

APiPost 是 API 设计、调试、文档、自动化测试的工具，下载地址 https://www.apipost.cn/

按照自己需求调试自己需要的格式和数据



接口测试结果

20.2 组态界面调试

功能说明

通过第一节的接口，调试自定义图表的展示需要的数据。

20.2.1 第一步 挑选自己需要的图表

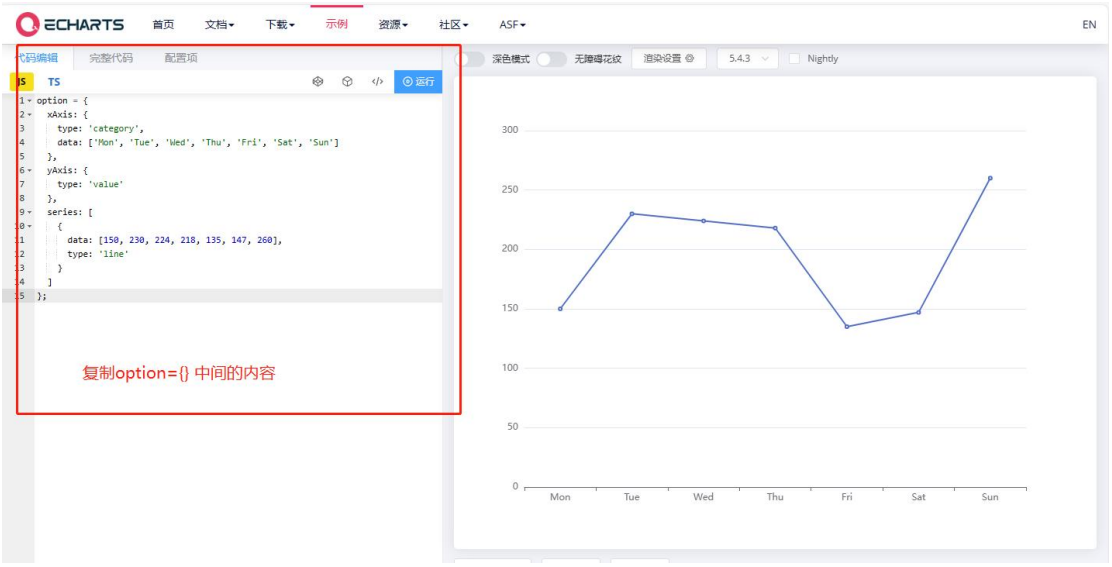
在 echarts 上面选择自己需要的图表，网址：

https://echarts.apache.org/examples/zh/index.html

基础折线图

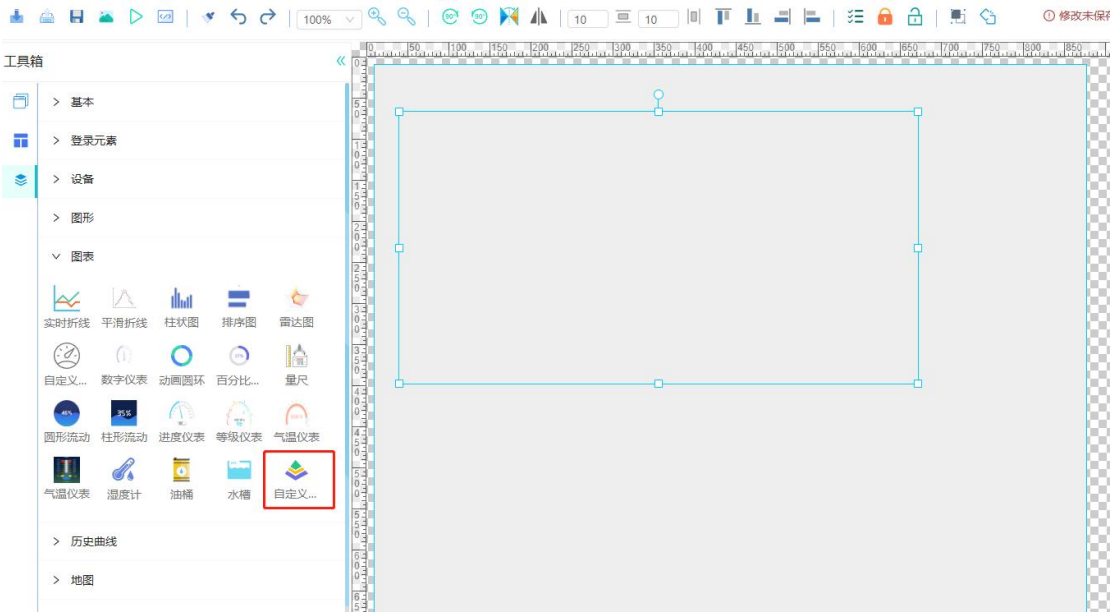
复制 option = {} 中间的内容

```
xAxis: {
    type: 'category',
    data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
},
yAxis: {
    type: 'value'
},
series: [
    {
        data: [150, 230, 224, 218, 135, 147, 260],
        type: 'line'
    }
]
```

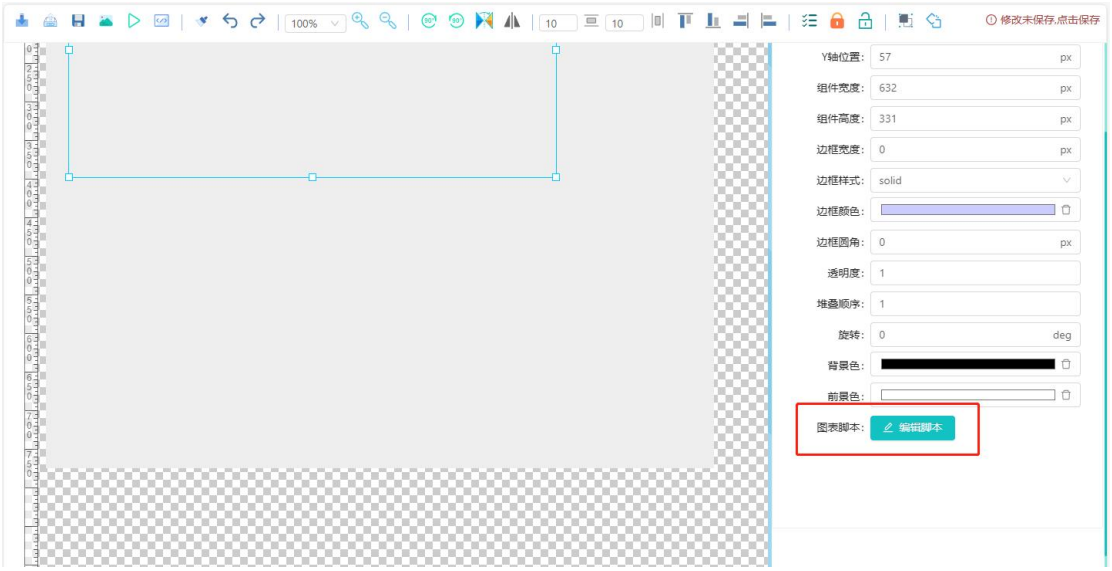


代码

20.2.2 第二步 在组态界面添加自定义图表



点击图表的编辑脚本按钮

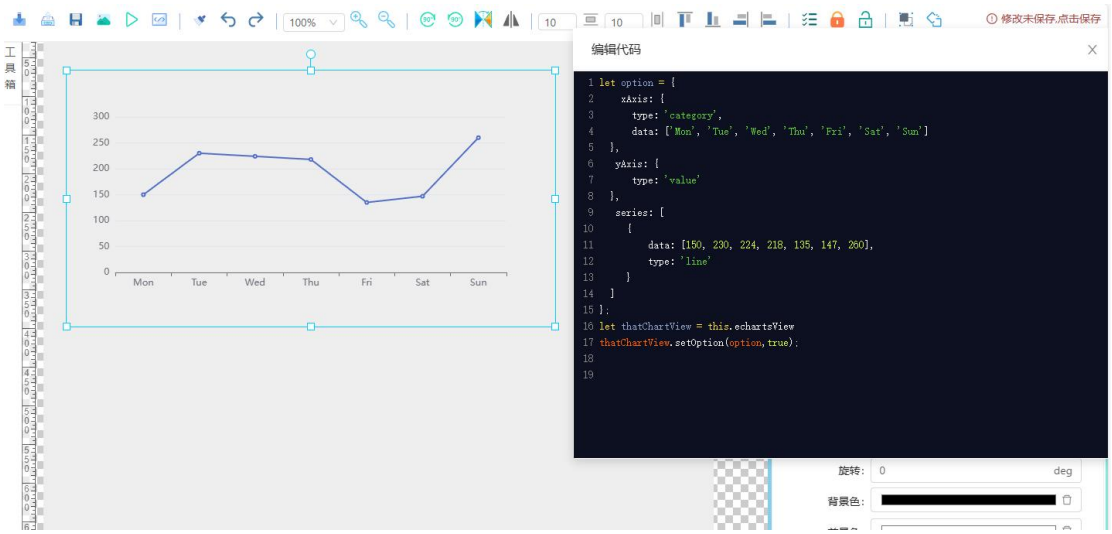


把刚才我们复制的代码粘贴到脚本里面，完整的代码如下

```
let option = {
    xAxis: {
      type: 'category',
      data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
  },
  yAxis: {
      type: 'value'
  },
  series: [
    {
      data: [150, 230, 224, 218, 135, 147, 260],
```

```
        type: 'line'
      }
  ]
};
let thatChartView = this.echartsView
thatChartView.setOption(option,true);
```

运行效果如下



20.3 让自定义的图表展示接口数据

功能说明

上节我们已经把把图表展示出来了，这节我们将说明，如何跟我们开发好的接口连接，展示我们的数据。

20.3.1 第一步 认识 ComponentRestApi 函数

ComponentRestApi 函数是请求远程 APi 接口数据，

ComponentRestApi(Type,URL,params).then(function (res){}).catch(function(e){})

参数	描述
Type	必需。要调用的函数或要执行的代码串
URL	必需。周期性执行或调用 code 之间的时间间隔，以毫秒计。
params	可选。 APi 请求的参数 json 对象
res	必需。 接口返回的数据
e	可选。 接口请求异常

例子参考

```
ComponentRestApi("Get","http://16.100.1.60:18089/ApiDemo",{}).then(function (res){
  //res 包含了接口返回的数据
  //比如我们刚才的接口返回的是
  //    "Code": 0,
  // "Msg": "成功",
  // "ResponseData": {
  //         "Register0": 0,
  //         "Register1": 0
  }
  //按照此代码就可以获取到接口返回的数值
  //res.ResponseData.Register0
  //.ResponseData.Register1

    //完整的接口数据信息
            console.log(res)
}).catch(function(){

    //这里是 API 接口异常捕捉
})
```

20.3.2 第二步 认识 setInterval 函数

如果我们想实时获取我们采集到数据，并通过图表展示出来，我们就要用到 setInterval 函数

setInterval() 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

提示：1000 毫秒= 1 秒。具体参照:https://baike.baidu.com/item/setInterval/2519822?fr=ge_ala

setInterval(code,millisec[,"lang"])

参数	描述
code	必需。要调用的函数或要执行的代码串
millisec	必需。周期性执行或调用 code 之间的时间间隔，以毫秒计。
lang	可选。 JScript 、 VBScript 、 JavaScript

```
//此段代码演示的就是 1 秒调用一次我们开发好的接口
setInterval(function(){
ComponentRestApi("Get","http://16.100.1.60:18089/ApiDemo",{}).then(function (res){

  }).catch(function(){

  })}, 1000);
```

20.3.3 第三步 使图表动起来

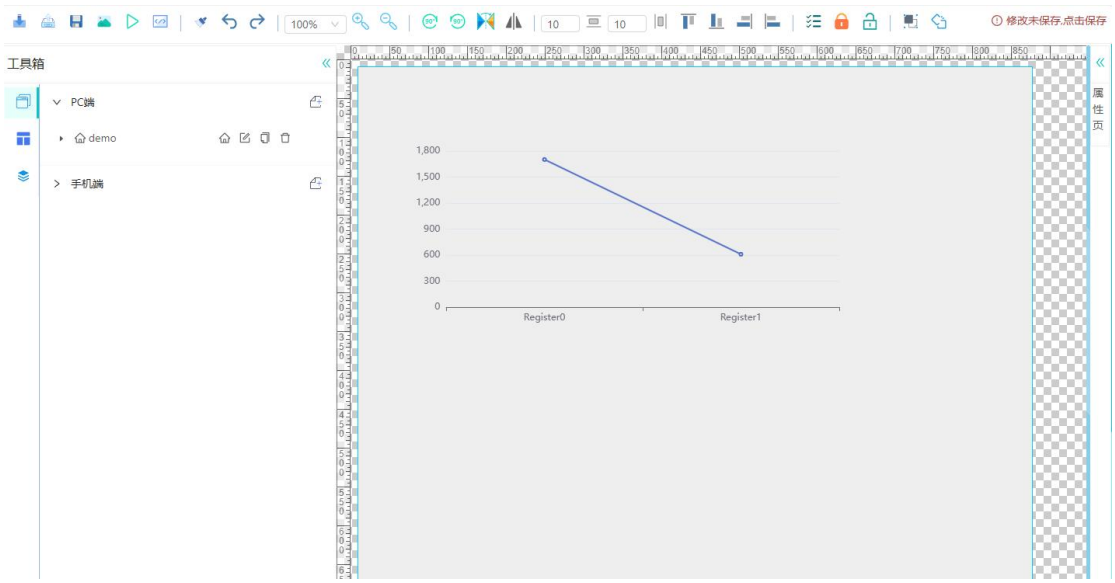
this.echartsView 是 echarts 初始化完成后的变量，用户可以调用 echarts 官网上的函数
this.echartsView.setOption(option,true);这里就是把 echarts 的图表选项重新绘制
注意：在定时器中不能使用 this 去指向，用户需要通过定义全局变量去指向此值

```
var thatEchartsView = this.echartsView;
thatEchartsView.setOption(option,true);
```

通过以前章节的积累，那么我们现在开始写接口和 echarts 结合的代码
完整的代码如下

```
//定义 eharts 的选项参数变量 let option = {
    xAxis: {
      type: 'category',
      data: ['Register0', 'Register1']
    },
    yAxis: {
      type: 'value'
    },
    series: [
      {
        data: [150, 230],
        type: 'line'
      }
    ]};//定义全局的 thatChartView 变量用于指向 echartsView 结构 let thatChartView = this.echartsView//初始化显示
thatChartView.setOption(option,true);
//开始周期性的调用开发的接口，并把接口返回的数据绘制到图表上 setInterval(function(){
    ComponentRestApi("Get","http://16.100.1.60:18089/ApiDemo",{}).then(function (res){
      //获取 Register0 的值
                              console.log(res.ResponseData.Register0)
      //获取 Register1 的值
      console.log(res.ResponseData.Register1)
      //把 Register0 和 Register1 的值赋值给 option.series[0].data 变量
      option.series[0].data = [res.ResponseData.Register0,res.ResponseData.Register1]
      //重新绘制图表
      thatChartView.setOption(option,true);}).catch(function(){
      //异常处理})}, 1000);
```

运行效果



20.4 关于 echarts

功能说明

从这几节说明可以看出，难点还是在 eharts 上，此节简述一下 echarts
复杂的图表可以在百度上搜索相关的教程

快速入门 Echarts

详细的介绍可以看这里
https://echarts.apache.org/handbook/zh/get-started/

最难理解的是 option 里面的参数设置，这里大家可以根据每个图表，查询相关的说明

https://echarts.apache.org/zh/option.html#title

教程发布抖音：Jxzdhw

官方微信：hweizdh

宏微自动化软件工作室

网址：http://www.hwzdh.cn/

授权购买：https://shop103372228.taobao.com/