

---

Update: 2025.12

# IOServer 使用说明

---

实时数据库数据网关



1 概述.....	1
1.1 和 rdbdac_ux 区别.....	1
1.2 系统组成.....	1
1.3 系统架构.....	2
1.4 运行环境.....	3
2 安装.....	4
2.1 安装 logsrv .....	4
2.1.1 windows 系统 .....	4
2.1.2 Linux 系统 .....	7
2.1.3 确认 logsrv 安装成功 .....	9
2.2 安装 ioserver.....	9
2.2.1 windows 系统 .....	10
2.2.2 linux 系统安装 .....	11
2.2.3 确认 ioserver 安装成功.....	12
2.2.4 ioserver 的默认账号.....	12
3 配置.....	13
3.1 ioserver 配置.....	13
3.1.1 双机热备冗余 .....	15
3.1.2 ioserver 服务.....	15
3.1.3 dblink 配置 .....	15
3.2 设备驱动配置.....	16
3.2.1 设备状态标签.....	19
3.2.2 MODBUS 标签表.....	19
3.2.3 OPCUA 标签表 .....	22
3.3 表达式计算.....	24
3.3.1 运算符.....	24
3.3.2 立即数.....	25

---

3.3.3 内置变量 .....	25
3.3.4 内置函数 .....	25
3.3.5 分支判断 .....	26
3.4 采集值工程值转换 .....	26
3.5 计算标签 .....	27
3.6 标签预处理 .....	27
3.6.1 fork 判断处理死区 .....	28
3.6.2 使用原 rdbdac_ux 模式 .....	28
4 维护管理 .....	29
4.1 登录 web 页面 .....	29
4.2 设备运行状态 .....	29
4.3 配置管理 .....	31
5 从 rdbdac_ux 迁移 .....	34
6 常见问题 .....	35
6.1 OPCDA 无数据 .....	35
6.2 OPCDA 设备启动失败 .....	35
6.3 设备无法启动 .....	36
6.4 WEB 页面出现乱码 .....	36

# 1 概述

从 2021.7 版本开始，rdb5 提供一个新的数据网关 ioserver，是一个现代的模块化的，可扩展的多进程数据采集控制输出网关，用于替代原 rdbdac\_ux 多线程架构的数据网关。

跨平台；支持 windows(x86\_64)和 Linux(x86\_64 和 aarch64)，支持 ARM 架构嵌入式平台。

故障隔离；设备驱动采用进程隔离，每个设备的驱动由一个独立的 IODevice 进程加载运行；其中一个驱动的故障不影响其他设备驱动和功能模块。

支持单个设备独立的管理控制。可在线更改设备配置，重启该设备生效，不影响其他正常运行的设备。

支持双机热备冗余部署，可整体故障迁移和设备故障迁移。

支持对上多实时库(热备冗余的库算一个库)连接，比如可同时连接中心库和本地库。对上连接也是独立进程模式，每个库的连接都是一个独立的 IOdblink 进程，互不影响。

提供 http/https 通道的 web 在线管理界面。

提供一个独立的日志服务器 logsrv，用于统一的日志写入存储服务 and 日志 WEB 页面在线查询浏览和下载服务。

## 1.1 和 rdbdac\_ux 区别

主要是架构的升级和功能的升级。

- 架构：由多线程模式改为多进程模式，实现模块的故障隔离。分为 4 类进程，logsrv 日志，IOServer 主进程，IODevice 设备驱动进程，IOdblink 实时库连接进程。
- 功能：架构的升级有些功能很容易实现。增加对单个设备的维护管理(配置下载提交，设备停止，启动)。增加多库连接，只需在配置文件里增加一个对上实时库的配置即可，而且连接的每个库都是双向，可对 ioserver 节点内的设备标签点做控制输出。更强大的表达式计算公式，支持数学函数和条件分支，增加计算标签表，支持设备之间的标签运算。

## 1.2 系统组成

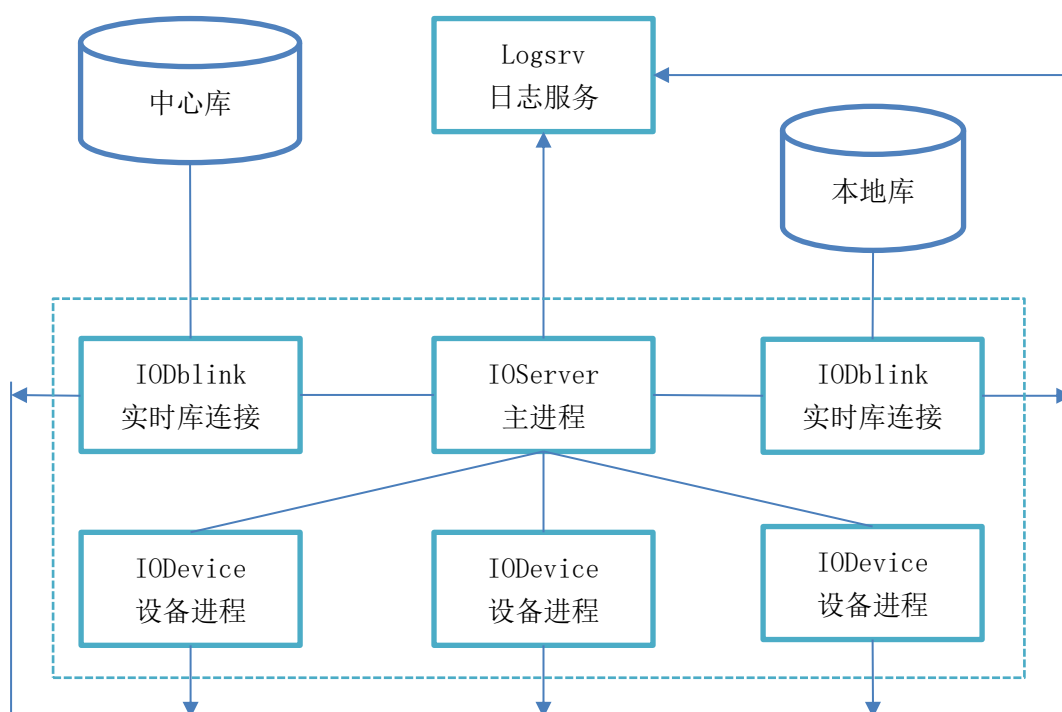
由 logsrv，IOServer，IODevice，IOdblink 4 部分组成。均提供 windows(x86\_64)和 linux(X86\_64 和 aarch64)版本。

- Logsrv；后台服务软件，独立的通用日志服务器，支持分仓和混仓，非本系统专用，支持 udp 通道写入，提供 http/https 在线查阅和下载服务。

- IOMServer; 台服务软件, 提供信息中心服务和对外管理维护服务。加载和监控其他模块进程后。
- IOMDevice; 加载设备驱动运行。
- IOMDbLink; 提供对实时库的连接和数据提交, 控制命令转发。
- 设备驱动, 一共有 5 个, ModbusRTU-TCP/485, OPCDA, OPCUA, OPCUAX(支持证书和加密通道), Simu(仿真驱动)。

## 1.3 系统架构

基于信息中心可扩展的进程间协作模式, 如下图。



- IOMServer, IOMDevice, IOMDbLink 部署在一台服务器或采集工作站机器内, 之间采用的是进程间通讯。其中 IOMServer 提供消息总线 and 对外管理维护服务。
- Logsrv 的日志写入基于 udp 通道, 可单独部署和其他应用公用或者部署在同一台机器内。
- IOMDbLink 到实时库走的协议可配置, 支持基于 ws/wss 的通道或者穿越网闸的单向 udp 协议。中心库或者本地库支持双机热备冗余部署。

---

## 1.4 运行环境

### 系统环境

#### Windows:

最低要求:Windows server2008,windows7 即以上 64 位系统。推荐 windows server 2016 x64 版。

#### Linux(X86\_64):

内核 3.10 及以上,libstdc++.so.6.0.19, glibc 2.17 及以上;典型系统 centos7.6; 推荐 centos8 和 ubuntu server1804

#### Linux(aarch64)

内核 4.4.131 及以上, libstdc++.so.6.0.21, glibc 2.23, 典型系统银河麒麟 V10 系统, 相当于 ubuntu16.04 的 aarch64 系统平台。

#### 硬件平台:

局域网环境最低要求: 4 核心 2G 以上频率主流 cpu; 8G 内存; 100M 网络带宽。根据所配置的设备多少和点数多少适当增加资源。

#### 云平台环境要求:

只推荐 linux 环境, 2 核心 2G 以上频率主流 cpu; 4G 内存; 网络带宽按照 20M/1 万标签点配置。

---

## 2 安装

规划和配置好运行环境，包括服务器，网络等。将软件复制到目标系统，执行相应的安装程序或脚本。

- 典型单机系统：一台服务器或者工作站机器安装 Logsrv 和 ioserver。
- 典型高可用系统：一台 logsrv 服务器，两台相同的 ioserver 服务器或工作站做高可用双机热备冗余，保证每个 ioserver 到 logsrv 的 udp 通道是通的并且没有被防火墙隔离，两台 ioserver 使用独立网口用心跳线直连。

### 2.1 安装 logsrv

Logsrv 是我公司开发的通用日志服务器，如果使用以前已经部署好的 logsrv 可以略过本步骤。

#### 2.1.1 windows 系统

将 logsrv\logsrv\_win64 整个目录拷贝到目标系统 C:\，用支持 utf-8 编码的纯文本编辑器打开 C:\logsrv\_win64\logsrv.ini 文件，看是否需要更改默认的配置。默认配置如下：

```
#日志配置-----  
  
[log]  
  
#日志服务协议, 目前仅支持 udp 协议  
  
protocol = udp://0.0.0.0:999  
  
protocol_ipv6 = udp://[::]:999  
  
#日志存储的跟目录  
  
path = c:/logsrv/log/  
  
#日志存储级别[err, wrn, msg, mor, dbg]之一  
  
level = dbg  
  
#日志存储模式, [mix, split] 文件存储模式, mix:混合存储; split:分开存储  
  
save_mode = split
```

---

#日志文件保存有效期, 0 表示长期有效, 大于 0 表示过期天数. 超期的将被删除

`valid_days = 30`

# max number files per cabin, >= 10, default 1000

`cabin_max_files = 100`

#web 服务配置, 用于查看下载日志, 不配置证书则使用 http-----

-----  
[webserver]

#服务协议

`protocol = http://0.0.0.0:999`

`protocol_ipv6 = http://[::]:999`

#根证书

`ca_root = ; root certificate file`

#服务器证书

`ca_server = ;c:/logsrv/ca/logsrv.cer ; server certificate file`

#服务器私钥

`ca_private = ;c:/logsrv/ca/logsrv.key ; server private key file`

#日志管理者

`username = admin ; web 登录账号`

`userpswd = admin ; web 登录密码`

如果不想提供本机以外的应用写入写入日志, 可以将[log]小结下的 `protocol` 改为  
`udp://127.0.0.1:999`

其他的可以更改日志的存储目录, 有效期, 每个目录最大文件存储数等。日志系统没有



单独的用户账号系统,只配置一个登录用户,为了安全,建议先改掉默认的用户账号和密码。

如果是部署在公网而且有 https 的服务器证书。建议配置服务器证书,以便支持 https 加密通道接入。证书的要求参见 ca.txt 文件描述,如果配置了证书,在[webserver]的 protocol 参数改为 <https://0.0.0.0:999> , 如果需要限制只能从本机的某个 ip 接入,可以将 0.0.0.0 改为具体的 ip 地址。

ca\_server:

服务器证书为带 SAN(Subject Alternative Name)扩展的 x509 v3 DER 格式证书。

ca\_root:

服务器根证书, 可选, 也要求为 x509 v3 DER 格式。

ca\_private:

服务器私钥, 注意保密, 不要泄露。文件为 PEM 格式。

确认配置无误后, 在 C:\logsrv\_win64 目录下打开管理员权限的 CMD 命令行或者 powershell 窗口。执行如下命令

```
./logsrv -install
```

安装为后台服务, 显示安装成功后, 到系统服务里启动 logsrv 服务, 如果安装不成功, 一般是没有管理员权限(注意看 cmd 窗口或者 powershell 窗口标题栏是否有“管理员”字样)。

安装好后在服务里如下图样子。



---

## 2.1.2Linux 系统

Logsrv 支持 x86\_64 和 aarch64 架构 CPU 的 linux 系统。下面以 x86\_64 平台为例，aarch64 安装方式完全一样。

logsrv 被安装为 systemd 的后台服务, 提供了一个 install.sh 脚本自动安装。同样要求 root 权限。

首先将 logsrv\_x86\_64\_inst 目录复制到目标系统任意目录。比如 ~/

**scp -r logsrv\_x86\_64\_inst [root@192.168.1.214:/home/](#)**

然后使用 vi 或者 vim 打开目标系统的/home/logsrv\_x86\_64\_inst/logsrv.ini 文件, 按照需求更改配置, 也可以改好了再拷贝过去。

配置和 windows 下基本相同, 唯一不同的是存储日志的目录格式。默认配置如下:

```
#日志配置-----
[log]

#日志服务协议, 目前仅支持 udp 协议
protocol = udp://0.0.0.0:999
protocol_ipv6 = udp://[::]:999

#日志存储的跟目录
path = /home/logsrv/

#日志存储级别[err, wrn, msg, mor, dbg]之一
level = dbg

#日志存储模式, [mix, split] 文件存储模式, mix:混合存储; split:分开存储
save_mode = split

#日志文件保存有效期, 0 表示长期有效, 大于 0 表示过期天数. 超期的将被删除
valid_days = 90
```

---

```
# max number files per cabin, >= 10, default 1000
```

```
cabin_max_files = 100
```

```
#web 服务配置, 用于查看下载日志, 不配置证书则使用 http-----
```

```
[webserver]
```

```
#服务协议
```

```
protocol = http://0.0.0.0:999
```

```
protocol_ipv6 = http://[::]:999
```

```
#根证书
```

```
ca_root = ; root certificate file
```

```
#服务器证书
```

```
ca_server = ; server certificate file
```

```
#服务器私钥
```

```
ca_private = ; server private key file
```

```
#日志管理者
```

```
username = admin ; web 登录账号
```

```
userpswd = admin ; web 登录密码
```

同样的可以配置证书和更改默认的服务接入 ip 和端口。记住改掉默认的日志管理者用户和密码。

配置确认无误后, 使用安装脚本安装为后台 systemd 服务。

在目标系统 `~/logsrvbin` 目录下打开终端, 使用

```
sudo ./install.sh
```

安装, 安装完成后会自动启动 logsrv 服务。

如果要停止，启动，或者查看状态，使用 `systemctl` 命令即可

停止服务：

```
sudo systemctl stop logsrv
```

启动服务：

```
sudo systemctl start logsrv
```

重启服务

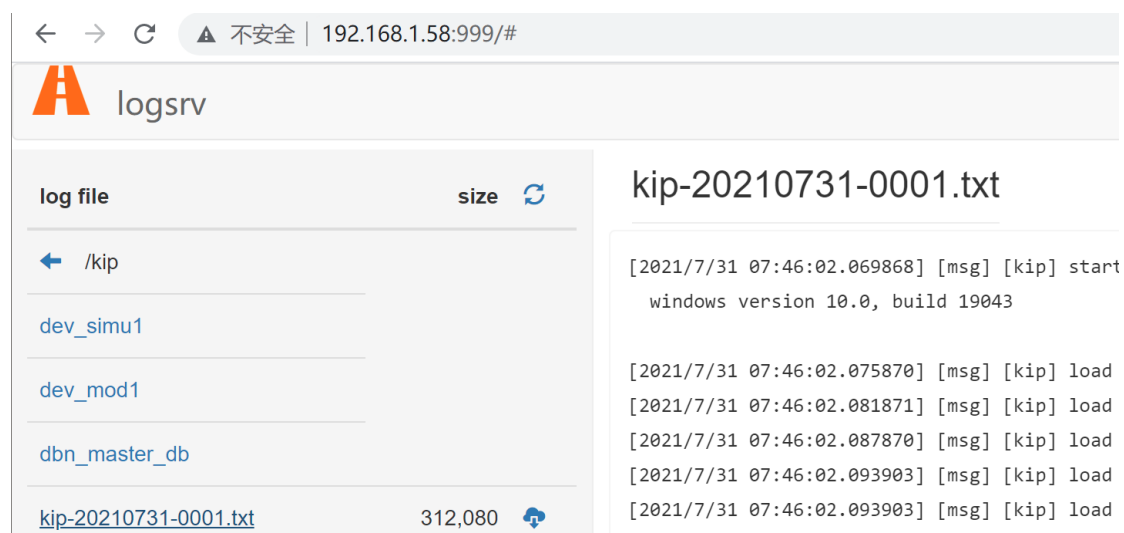
```
sudo systemctl restart logsrv
```

如果需要了解更多安装过程，请打开 `install.sh` 阅读脚本内容。Logsrv 被安装到 `/usr/local/bin/logsrv` 目录下。

## 2.1.3 确认 logsrv 安装成功

启动 logsrv 成功后，使用支持 html5 的浏览器，edge, chrome, firefox 等，地址栏输入 logsrv 的服务器 url 看是否能成功登录并显示日志。默认端口 999，例如：

`http://192.168.1.214:999`

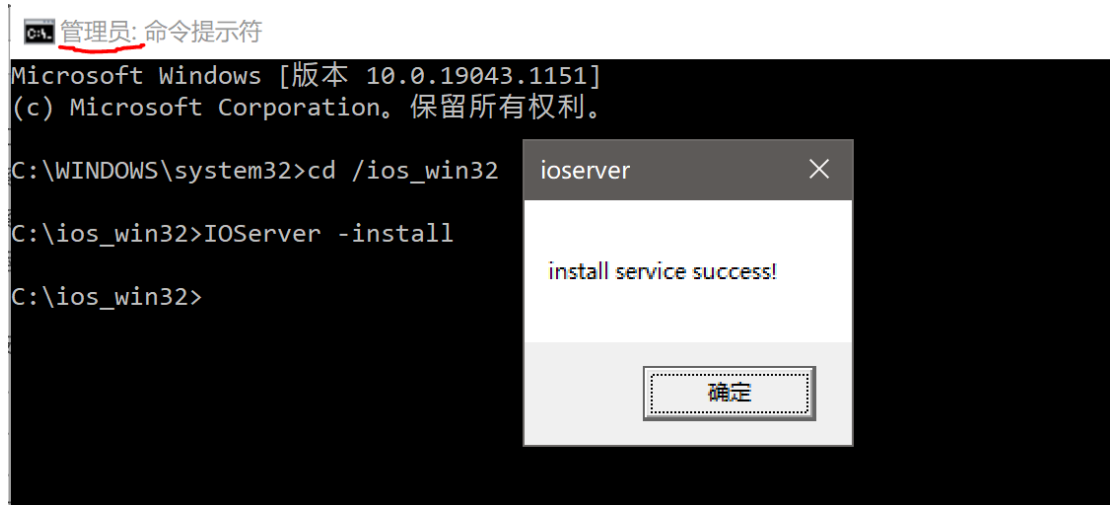


## 2.2 安装 ioserver

驱动程序和配置文件分别放在 `drivers` 和 `config` 子目录下，可以先配置好后在安装到目标系统，驱动设备的配置也可以安装后使用 web 页面在线配置。

## 2.2.1 windows 系统

将 ioserver/ios\_win32 整个目录配置到目标系统 C:/下；



用支持 utf-8 编码的纯文本文件编辑器（例如 windows 10 的 notepad 或者第三方的 notepad++）打开 config 子目录下的配置文件，配置（具体配置方法参见后买你的第 3 章 配置 ioserver）好后进行安装。在 C:/ios\_win32 目录下打开管理员权限下的 CMD 窗口或者 powershell 窗口。执行 `IOServer -install`

显示安装成功后，到服务里查看并启动服务，如果启动不成功，检查主要配置是否正确，成功后查阅日志，看各个模块是否都成功加载并启动。



## 2.2.2 linux 系统安装

提供 x86\_64 和 aarch64 架构 CPU 的 linux 平台安装包，和 windows 下安装类似，提供了一个安装脚本 install.sh，下面以 x86\_64 平台为例描述安装过程。

使用 windows linux 子系统先把 ios\_linux\_x86\_64 整个目录复制到目标系统某个路径，比如 /home 目录，如下例子：

```
scp -r ios_linux_x86_64 root@192.168.1.214:/home
```

---

然后使用 windows linux 子系统或者 putty 用 root 权限的账号登录到目标系统， cd 到/home/ios\_linux\_x86\_64 目录下执行 install.sh

```
sudo ./install.sh
```

安装成功后提示 “install success”，并没有启动服务，提示使用 systemctl 启动服务。

最终会安装到 /usr/local/bin/ioserver 目录下，如果要手工修改配置，请使用 vi 命令打开/usr/local/bin/ioserver/config/io-server.ini

在配置检查正确无误后，使用

```
sudo systemctl start ioserver.service
```

 启动服务。

以后可以使用 systemctl 的 start, stop, restart, 等操作 ioserver 服务，在 systemctl 命令里，ioserver.service 可以简写为 ioserver

## 2.2.3 确认 ioserver 安装成功

启动 ioserver 成功后，使用支持 html5 的浏览器，edge, chrome, firefox 等，地址栏输入 ioserver 的服务器 url 看是否能成功登录并进入管理界面。

## 2.2.4 ioserver 的默认账号

ioserver 内置了三个账号，登录后可以在 web 页面的配置管理页面修改当前登录者自己的密码。

账号	默认密码	权限	说明
admin	ioserver	所有权限	管理员，用户和角色管理
engineer	engineer	上传下载配置, 控制设备启停	工程师
operator	operator	控制设备启停	值班员

目前暂未提供 admin 账号的用户和角色管理界面。

---

## 3 配置

所有的配置放在 config 子目录中。包含三部分：ioserver 配置，dblink 配置和设备配置。

### 3.1ioserver 配置

包含日志，ioserver 对外服务，双机热备冗余，dblink 连接配置。使用支持 utf-8 编码的纯文本文件编辑器打开 io-server.ini 文件，默认的配置如下：

#日志配置

[log]

#日志服务器

log\_url = udp://127.0.0.1:999/io

#日志输出级别 [err,wrn,msg,mor,dbg]

log\_level = dbg

#服务器配置

[service]

#节点名,ioserver 的引用名，全局唯一

nodename = ioserver1

#管理服务协议，http 表示开启 PB3 和 http/websocket；https 表示启用 TLS1.2 加密通道上的 pb3 和 http

protocol\_srv = http://0.0.0.0:923

protocol\_srvip6 = http://[::]:923

#根证书,可以不填,x509 v3 DER file

ca\_root =



---

```
# 服务器证书,x509 v3 DER file

ca_server =

# 服务器私钥, PEM 格式

ca_private =

# 本机进程间通讯服务协议, windows 将使用 IF_UNIX 地址
"tcp://127.0.0.191:805", unix 将使用 AF_UNIX 地址 "/var/tmp/ipc:805"

protocol_ipc = ipc://127.0.0.191:805

# 高可用双机冗余心跳线协议配置, 主机的协议为 ha_master, 从机的协议为
ha_slave

#protocol_ha = ha_master://192.168.1.56:959/io_heart
#如果是从机则配置如下:
#protocol_ha = ha_slave://192.168.1.58:959/io_heart

#对上提交 dblink 配置, 可以有 0-N 个[dblink]小结, 每一个小结 name。

[dblink]

#实时库引用名, 唯一。

name = master_db

#多链接参数, 支持故障切换。内容使用 JSON 格式

config = io-dblink.json

#日志级别

loglevel = dbg
```

---

#结束

[end]

如果 logsrv 不在本机，需要将[logsrv]中 protocol 的 ip 地址改为正确的 IP 地址。其他配置可以采用默认值不改动。

### 3.1.1 双机热备冗余

默认没有开启双机热备，如果有高可用需求，需要配置 protocol\_ha 协议。主机和备机的 ip 端口和心跳名必须完全相同，主机协议为 ha\_master，从机协议为 ha\_slave，例子参考配置之注释掉的部分。一般心跳线采用直接连接，独立的网口和 ip。

### 3.1.2 ioserver 服务

在[service]小结，protocol\_srv 提供管理服务，基于 http/https，默认是 http 协议。如果配置了服务器证书（证书的格式和要求和前面的 logsrv 的服务器整数相同），要改成 https 协议。默认使用 923 端口绑定所有 ip，可以改为绑定具体 IP。

protocol\_ipc 提供了进程间通信协议，使用默认配置即可。

### 3.1.3 dblink 配置

对实时库的连接配置在 io-server.ini 文件的[dblink]中，一个[dblink]小结表示一个实时库连接。是 IODblink 进程使用的配置，一个 ioserver 可以向多个实时库节点(主备算一个节点)提交数据，每个 IODblink 进程负责提交一个节点。

name: 全局唯一的实时库引用名。默认库名为 master\_db，也可以改成其他名字。比如 center\_db，不建议更改，因为使用 msater\_db 名可以以兼容方式在 rdbman 的工具里看到该节点的对上实时库链接状态信息。如果再加一个实时库同理建议引用名为 slave\_db。

config: 实时库的连接参数配置文件，可能是双机热备的实时库，一个库对应一个配置文件，默认库的配置文件为 io-dblink.json，库配置文件名无要求只要能区分开。如果是冗余库，可以配置两个连接参数。单库只需配置一个连接参数，io-dblink.json 默认按照冗余库配置：

```
{  
  "dblinks": [{  
    "node": "master",  
    "url": "ws://192.168.1.201:921",  
    "srvcafile": "",
```

---

```

        "user": "admin",
        "pswd": "admin"
    }, {
        "node": "slave",
        "url": "ws://192.168.1.69:921",
        "srvcafile": "",
        "user": "admin",
        "pswd": "admin"
    }
] ,
"cache_path": "/home/ioscache/db1/",
"cache_maxsize": 4000,
"cache_maxhour": 8
}

```

如果走加密通道, url 为 wss 协议, 需要配置服务器证书 srvcafile, 证书格式同 logsrv 的服务器证书。

如果是走单向网闸, url 配置为 gapudp 协议:

```
"url": "gapudp://192.168.1.69:921",
```

**cache\_path** 字段是本地缓存目录, 自 **rdb2023.4**(内部版本 **5105**)开始提供, 不配置表示不开启本地缓存。当网络故障或者实时库维护无法连接到实时库时(**dblinks** 字段提供的主备都不通), **ioserver** 中的 **IODblink** 进程会将数据暂存到指定的缓存目录, 待故障恢复后采用历史补录方式提交到实时库。

**Cache\_maxsize** 字段表示本地缓存最大容量, 单位 **MB**, 大于该限制后自动删除最旧的, 循环利用空间, 默认值 **4000Mb**, 自 **rdb2023.5**(内部版本 **5106**)开始提供此功能。

**Cache\_maxhour** 字段为本地缓存时间限制, 最大小时数, 该小时之前的旧数据自动删除, 默认值 **8** 小时, 自 **rdb2023.5**(内部版本 **5106**)开始提供此功能。

## 3.2 设备驱动配置

设备的驱动程序放在 **drivers** 目录里, 是动态库, 兼容原 **rdbdac\_ux** 的驱动, windows 下为 32 位的 **dll** 文件, linux 为 64 位的 **so** 文件。目前提供有 **opcda**(仅 Windows 版), **opcua**(不支持证书和安全策略), **opcua**(支持证书和安全策略), **modbus-rtu for tcp/485** 和

---

一个仿真驱动 `simudrv`。

一个驱动程序可以配置多个驱动实例去连接一个设备。在 `io-device.json` 中配置具体的设备。作为例子，默认为每个驱动配置了一个设备。

```
{
  "device": [{
    "name": "mod1",
    "driver": "dac_modrtu",
    "config": "mod_cfg.ini",
    "expfile": "mod_exp.csv",
    "loglevel": "dbg",
    "autoload": "yes",
    "snaptime": "driver",
    "forcesubmit": 0
  }, {
    "name": "simu1",
    "driver": "simudrv",
    "config": "simu1.ini",
    "expfile": "",
    "loglevel": "dbg",
    "autoload": "yes",
    "snaptime": "driver",
    "forcesubmit": 0
  }, {
    "name": "ua1",
    "driver": "dac_opcua",
    "config": "opcua1.conf",
    "expfile": "",
    "loglevel": "dbg",
    "autoload": "yes",
```

---

```

        "snaptime": "driver",
        "forcesubmit": 0
    }, {
        "name": "da1",
        "driver": "dac_opcdrv",
        "config": "opc1.conf",
        "expfile": "",
        "loglevel": "dbg",
        "autoload": "yes",
        "snaptime": "driver",
        "forcesubmit": 0
    }
]
}

```

最大配置多少个设备并没有从软件角度做限制，具体取决与 CPU，内存，网络资源是否足够，每个设备有 7 个参数需要配置。

**name**：唯一的设备引用名。

**driver**：驱动程序名，不带路径，不带文件后缀名。

**config**：每个驱动自己的配置文件，不带路径放在 config 目录下，不同的驱动不同。

**expfile**：输入输出表达式转换表文件。配置需要做采集值和工程值需要转换的标签的转换公式。有些驱动比如 opcua 本身就是工程值无需转换，可以配置空或不要 expfile 字段。后面具体描述表达式转换。

**loglevel**：日志输出级别，err, wrn, err, mor, dbg 之一。

**autoload**：启动时加载并启动设备。

**snaptime**：时标源，driver, local, rdb 之一。分别表示使用驱动时标，本地时标和实时库时标。

**forcesubmit**：强制提交间隔秒数(2022-11 版起增加的功能)；默认值 0 表示关闭强制提交，5-1000 表示强制提交的间隔秒数；解决有些驱动采用订阅模式数据不变化不更新的数据提交问题，比如 OPCDA/OPCUA，当一个标签点值长期不变时，OPC server 不会再推送数据，此时如果将 forcesubmit 配置为 10，且和 opcserver 连接正常，上一个值得质量是 good，则 ioserver 会每隔 10 秒自动更新数据时标(值和数据质量不变)，并向实时库提交数据，这

样在实时库中看到该标签虽然值未变,但是时标没 10 秒会更新一次,如果配置有趋势压缩,并不会增加实时库得落地样本值记录数。

第三方的设备驱动动态库放入 drivers 目录,如果第三方的设备驱动有其它依赖的动态库,需要把依赖库放到和 iosever 的根目录(即和 rdbapi.dll 或 librdapix64.so 相同的目录)。Windows 和 linux 都是同样要求。

### 3.2.1 设备状态标签

每个驱动设备会自动生成一个设备状态标签,命名规则为:

Iosever 节点名.io\_设备名.st

在导出该设备标签时会被自动导出。比如例子中的设备 4 个设备会导出以下 4 个标签:

**iosever1.io\_mod1.st**

**iosever1.io\_simu1.st**

**iosever1.io\_ua1.st**

**iosever1.io\_da1.st**

状态标签的类型为 DT\_DIGITAL,取值范围[0,3]

0: 正常;

1: 禁用;

2: 停止;

3: 故障;

这些设备状态标签被导入实时库,会被 Iosever 周期性的更新,可以通过监控设备状态标签来监控设备运行状态。

### 3.2.2MODBUS 标签表

按照 MODBUS 的规范,数据按照寄存器组织,有 1bit 寄存器和 16bit 的寄存器,16bit 也叫 WORD 或中文的“字”,大于一个 WORD 的数据采用连续的多个 WORD 寄存器存储,字节顺序约定为大头格式 (big-Endian),即高字节在前(低地址)。

寄存器地址约定,采用 MODBUS MASTER 协议地址,使用 6 位 10 进制数表示地址:

表 3-2 MODBUS 寄存器地址定义表

寄存器地址	说明
<b>1-65536</b>	<b>Coils</b> 输入: 对应 MODBUS 命令 0x01 地址 0-0xFFFF 输出: 对应 MODBUS 命令 0x05/0x0F 地址 0-0xFFFF

100001-165536	<b>Discrete Inputs</b> 输入：对应 MODBUS 命令 0x02 地址 0-0xFFFF
300001-365536	<b>Input Registers</b> 输入：对应 MODBUS 命令 0x04 地址 0-0xFFFF
400001-465536	<b>Holding Registers</b> 输入：对应 MODBUS 命令 0x03 地址 0-0xFFFF 输出：对应 MODBUS 命令 0x06/0x10 地址 0-0xFFFF

为了便于配置,本系统约定 MODBUS 的数据类型如下:配置是也可以填写括号里的类型,比如 i2 等价于 int16

表 3-3 MODBUS 数据类型定义表

MODBUS 数据类型	描述
bit	1 位, MODBUS MASTER 协议寄存器地址为 1-65536 和 100000-165536 使用。
i2(int16)	16 位带符号整数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 一个地址
ui2(uint16)	16 位无符号整数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 一个地址
i4(int32)	32 位带符号整数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 连续两个地址
ui4(uint32)	32 位无符号整数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 连续两个地址
f4(float)	32 位 IEEE754 浮点数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 连续两个地址
f8(double)	64 位 IEEE754 浮点数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 连续四个地址
i8(int64)	64 位带符号整数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 连续四个地址
ui8(uint64)	64 位无符号整数,占 MODBUS MASTER 协议寄存器地址为 300000-365536 和 400000-465536 连续四个地址

标签表的配置参见例子,其中 A1 单元格为文件格式标识字符串“modbustags”,不能更改,否则 dac\_modrtu.dll 驱动不认。

表 3-4 MODBUS 标签表属性定义:

unitID	设备地址,总线上唯一。
regAddr	寄存器地址,6 位 10 进制表示法。

<b>dataType</b>	MODBUS 设备的数据类型。
<b>rdbTagName</b>	实时库表签名
<b>rdbDT</b>	实时库数据类型
<b>engUnits</b>	工程单位
<b>description</b>	描述
<b>byteorder</b>	<p>4 字节和 8 字节数据的字节顺序(从 2020.7 版开始两字节的数据也支持位置交换), 不填写为默认模式, 填写 <b>inverse</b> 表示 <b>big-endian</b> 模式(即 <b>modbus_slave</b> 工具中的 <b>float_inverse</b>, <b>long_inverse</b> 等)</p> <p><b>float</b> 为例:</p> <p>默认模式: LH LL HH HL 或 Modbus Slave( CD AB )</p> <p><b>inverse</b> 模式: HH HL LH LL 或 Modbus Slave( AB CD)</p> <p>从 2021.11 版开始, 增加下面 6 种模式:</p> <p><b>WORD</b> 两种模式:</p> <p>12 : LH; 小头 WORD, 等于原 <b>inverse</b> 模式</p> <p>21 : HL; 大头 WORD, 原默认模式</p> <p>双 WORD, 4 种模式: 适合 <b>int32</b>, <b>float32</b></p> <p>1234 : LL LH HL HH; 4 字节</p> <p>2143 : LH LL HH HL; 原默认模式或 Modbus Slave( CD AB )</p> <p>4321 : HH HL LH LL; 原 <b>inverse</b> 模式或 Modbus Slave( AB CD)</p> <p>3412 : HL HH LL LH;</p> <p>四 WORD, 4 种模式: 适合 <b>int64</b>, <b>float64</b></p> <p>1234: DWL(LL LH HL HH) DWH(LL LH HL HH), 即字节顺序为“12345678”;</p> <p>2143 : DWL(LH LL HH HL) DWH(LH LL HH HL) , 即字节顺序为“21436587”; 原默认模式或 Modbus Slave( GH EF CD AB )</p> <p>4321: DWH(HH HL LH LL) DWL(HH HL LH LL) , 即字节顺序为“87654321”; 原 <b>inverse</b> 模式或 Modbus Slave( AB CD EF GH)</p> <p>3412: DWH(HL HH LL LH) DWL(HL HH LL LH) , 即字节顺序为 “78563412” ;</p>

MODBUS 的例子标签表如下图:

Linux 和 windows 版的标签表是完全一样的。



从 2020.7 版开始, modbus 驱动支持驱动级别的位分解。寄存器后面用小数点分开后跟 bit 位, 0 表示 bit0, 比如 300050.0 表示 bit0 位, 300050.3 表示 bit3 位, modbus 类型填写 bit, 实时库类型填写 digital, 如下

```
1,300050.0,bit,mod1.bit5000,digital,,300050
```

```
1,300050.1,bit,mod1.bit5001,digital,,300050
```

经过以上配置后, rdbdac\_ux 会自动在 MODBUS 的数据类型和实时库数据类型之间转换。但是, 很多 MODBUS 设备采用定点数据约点小数点方式表示浮点数。比如 MODBUS 使用 16 位整数 (即一个字) 值 12345, 来表示小数 123. 45, 这是通过以上标签表就不能正确转换, 还需要配合计算表达式来转换, 具体参见《ioserver 网关使用说明. pdf》中的采集值工程值转换。

### 3. 2. 30PCUA 标签表

标签表和 OPCDA 差不多, 多一列 nsindex(命名空间索引, 是一个 0-65535 的值), opc 标签名直接使用 nodeid 配置。

dac\_opcua 和 dac\_opcuax(支持证书, 具体参见《OPCUA 驱动使用说明.pdf》)两个版本的标签表是完全一样的。

表 3-5 OPCUA 标签表字段属性定义

rdbtagname	实时库标签名
nsindex	opcua server 中标签的命名空间索引(namespaceIndex), 从 OPCUA server 那里获取
nodeid	opcua server 标签的 nodeid(node identifier)。默认为字符串, 从 rdb2020.8 开始支持数字, "i="开头为数字型, "s="开头为字符串型。例如下面三个都是支持的: i=10032 s=m1.intval2 m1.intval2
datatype	数据类型: Digital 开关量 Int32 4 字节整数 Float32 单精度浮点型, 4 字节 Int64 8 字节整数 Float64 双精度浮点型, 8 字节 String 字符串型 -----

	<p>下面是 <b>rdb2022.7</b> 开始支持分别配置实时库类型和 <b>opcua</b> 类型；用于某些 <b>opcua server</b> 下行控制数据类型必须相同的场景格式</p> <p>实时库类型:<b>opcua</b> 类型</p> <p>中间是西文冒号分开，前面是实时库类型，后面是 <b>opcua</b> 类型。如果实时库类型和 <b>opcua</b> 类型完全相同，只填写实时库类型，不要冒号和 <b>opcua</b> 类型(兼容 <b>rdb2022.7</b> 以前版本)。</p> <p><b>opcua</b> 特有类型定义如下：</p> <p><b>bool</b> 1 字节 <b>boolean</b> 类型，自动转为实时库 <b>digital</b></p> <p><b>char</b> 1 字节整数，自动转换为实时库 <b>int32</b></p> <p><b>byte</b> 1 字节无符号整数，自动转换为实时库 <b>int32</b></p> <p><b>short</b> 2 字节整数，自动转换为实时库 <b>int32</b></p> <p><b>word</b> 2 字节无符号整数，自动转换为实时库 <b>int32</b></p> <p><b>dword</b> 4 字节无符号整数，自动转换为实时库 <b>int32</b></p> <p><b>uint64</b> 8 字节无符号整数，自动转换为实时库 <b>int64</b></p> <p>例子 1: 实时库 <b>digital</b>, <b>opcua</b> 是 <b>boolean</b>  <b>digital:bool</b></p> <p>例子 2: 实时库 <b>digital</b>, <b>opcua</b> 是 <b>char</b>  <b>digital:char</b></p> <p>例子 3: 实时库是 <b>float32</b>, <b>opcua</b> 是 <b>word</b>  <b>float32:word</b></p> <p>例子 4: 实时库是 <b>int32</b>, <b>opcua</b> 是 <b>short</b>  <b>int32:short</b></p> <p>注意：如果是只读标签，不用配置 <b>opcua</b> 类型，<b>IOServer</b> 会自动转换，按照以前版本配置即可。</p> <p>只有 <b>rw</b> 标签，需要向下控制输出的，才需要精确配置 <b>opcua</b> 类型，以便不支持自动转换的 <b>opcua server</b> 产生 <b>0x80740000</b>(类型不匹配)错误。其中 <b>bool</b> 转换是 <b>dac_opcdax</b> 驱动 <b>1.0.6</b> 版支持(<b>rdb2025.12</b>)。</p>
<b>access</b>	访问方式, <b>r</b> 表示读, <b>rw</b> 表示读写, 带有 <b>w</b> 的表示可以写, 会被 <b>rdbdac_ux</b> 注册到实时库当作可 <b>device</b> 写方式的标签, 即控制标签, 其他客户端如果有写 <b>device</b> 权限, 即控制权限, 就可向这个设备发送下传控制数据。
<b>Unit</b>	工程单位。
<b>Description</b>	描述

具体参见 `/rdbdac_ux/opcuatag.csv` 中的配置。注意 csv 文件的第一行和第二行不要改动。

---

## 3.3 表达式计算

计算标签和工程值-采集值转换采用计算公式实现,公式支持内置函数,支持判断分支,比 rdbdac\_ux 公式转换强大很多。

### 3.3.1 运算符

运算符:

|| 逻辑或

&& 逻辑与

| 位或

^ 位异或

& 位与

== 等于

!= 不等于

< 小于

<= 小于等于

> 大于

>= 大于等于

<< 左移位

>> 右移位

+ 加

---

- 减

\* 乘

/ 除

% 除模

- (负号)

~ 位反

! 逻辑反

优先级和结合率同 C 语法。

### 3.3.2 立即数

int : 包含有符号, 无符号, 32 位 63 位, 比如 100, 0x64

double : 比如 12.5, 100.0

string : 西文双引号标记的字符串, 比如 "fork"

### 3.3.3 内置变量

Val : 当前标签值, 仅转换时有效

Qa : 当前标签数据质量, 仅转换时有效

### 3.3.4 内置函数

包含常用的数字转换和数学运算函数, 以后会扩充更多, 函数名不分大小写:

```
double atof (const char* str);
```

```
int atoi (const char * str);
```

```
double fabs (double x);
```

```
int abs (int x);
```

```
double sqrt (double x);
```

```
double sin(double x);
```

---

```
double cos (double x);  
double tan (double x);  
double log (double x);  
double log10 (double x);  
double ceil (double x);  
double floor (double x);  
  
val pv(const char* tagname); //取标签的工程值，仅计算标签中有效  
val qv(const char* tagname); //取标签的数据质量，仅计算标签中有效
```

### 3.3.5 分支判断

表达是一行，无法用传统的 if else 来实现，采用内置的 fork 函数来执行条件选择，可多层嵌套实现多路分支。Fork 函数有三个参数：

Expif: 条件表达式，结果为 0 或非 0

Exptrue: 真表达式；当 expif 结果不为 0 时被执行。

Expfalse: 假表达式；当 expif 结果为 0 时被执行。

Retval Fork(Expif, Exptrue, Expfalse)

Fork 返回值: Expif 不为 0，返回 Exptrue 的值，否则返回 Expfalse 的值。

举个例子说明。

```
fork(pv("tag_uint32") & 0x01, 1, 0)
```

```
fork(pv("tag_uint32") << 1, 1, 0)
```

```
fork(pv("tag_uint32") << 2, 1, 0)
```

这是一个整数转开关量的例子，分别将 tag\_uint32 标签工程值的 bit0, bit1, bit2 转换为 3 个开关量。

## 3.4 采集值工程值转换

每个设备配置一个表达式转换表文件，csv 格式，注意一定要使用 excel 编辑后导出为 csv 文件，不要手工编辑，因为公式里面可能有字符串的双引号，需要做转码处理，手工编写可能会出错。csv 文件存放在 config 目录，配置到每个设备的 expfile 字段中。

Expin 为输入转换，采集值转换为工程值表达式。

Expout 为输出转换，工程值转换为才机制表达式。

下面最简单的例子，mod1.i2tof4rw 采集值为两位小数的定数。

C7			
	A	B	C
1	ioexpression		
2	TagName	expin	expout
3	mod1.i2tof4rw	val/100.0	val*100
4			
5			
6			
7			
8			

支持内置变量 标签值 val 和数据质量 qa

### 3.5 计算标签

计算标签不在任何设备的标签表中。可以实现汇总，数据分解(一个设备标签分解为多个实时库标签)。典型应用场景：

- 统计标签：将从设备 1，设备 2，设备 3 来的三个标签相加生成一个实时库统计标签。比如机组 1，机组 2，机组 3 的总功率。
- 数据分解：将 modbus 一个寄存器表示的 16 个状态量分解为实时库中的 16 个状态量标签。

计算标签表文件整个 ioserver 系统只有一个, 固定为 config/io-calc.csv 文件。

E8						
	A	B	C	D	E	F
1	kip-calc					
2	TagName	DataType	Unit	Description	OPCUA	Expression
3	tag_k1	DIGITAL		bit1	1	fork(pv("tag_uint32")<<1,1,0)
4	tag_sum	FLOAT64	kg	sum	1	pv("tag_float")+pv("tag_int32")
5						

计算标签使用 **pv()** 函数来获取引用的标签值，使用 **qv()** 来获取引用的标签质量。

### 3.6 标签预处理

可以采用原来 rdbdac\_ux 的预处理方式，不分设备，ioserver 只有一个配置文件 **predotags.csv**；也可以采用新方法用 **fork** 来实现。推荐在每个设备的采集值工程值转

---

换中使用 **fork** 分支来实现。

### 3.6.1 fork 判断处理死区

在每个设备的 expfile 指向的 csv 文件中，输入表达式公式中使用 fork 来过滤掉死区值，比如采集值 0-5v 对应工程值 0.0-50.0，设备是 modbus 设备，寄存器存储的是 0-5000 表示 0-5v，假设采集值小于 5 的视为死区，强制为 0，则表达式可以向像下面这样写：

```
Fork(val<5, 0, val/100.0)
```

这样小于 5 时返回 0；大于 5 返回 val/100.0

如果还有上限死区(大于 5000 的按照 5000 计算)，可以嵌套使用 fork

```
Fork(val<5, 0, fork(val > 5000, 50.0, val/100.0))
```

### 3.6.2 使用原 rdbdac\_ux 模式

配置一个预处理 csv 文件，和原来的 rdbdac\_ux 一样，只是文件名必须是固定的 **config/predotags.csv**。

## 4 维护管理

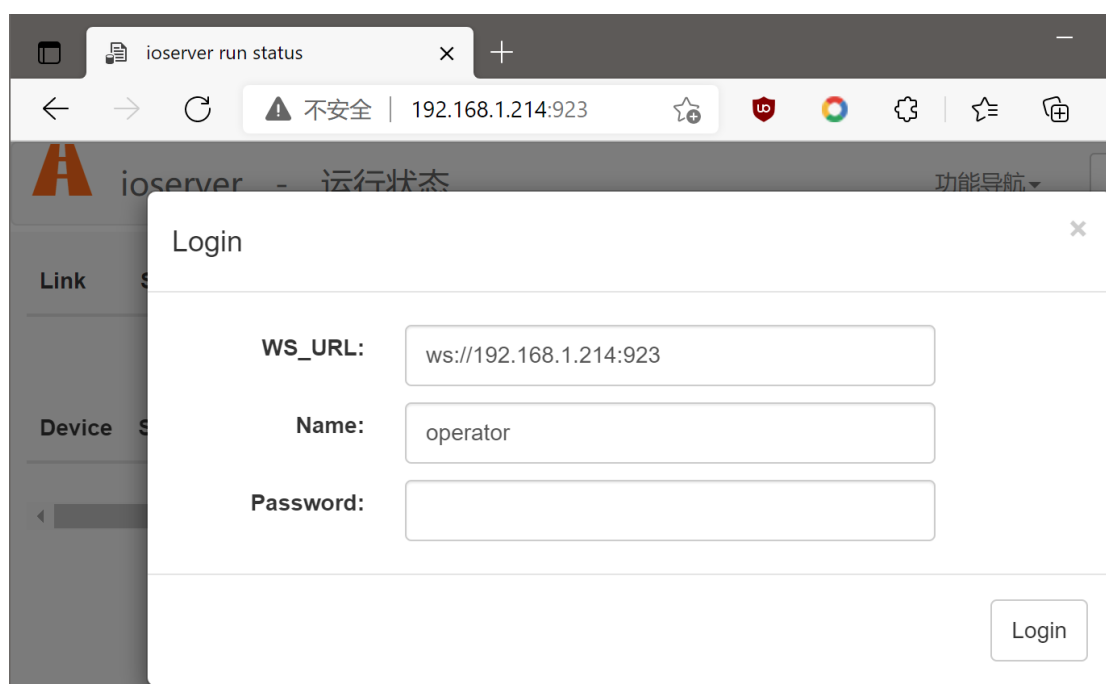
ioserver 支持通过 web 直接管理维护，大部分的操作都能完成，包括：设备启动/停止，设备配置文件的上传下载，导出标签表，更改当前登录用户密码，查看当前对实时库的连接状态，每个设备的运行状态，标签快照数据。

### 4.1 登录 web 页面

使用支持 html5 的浏览器，edge，chrome，firefox 等，在地址栏输入 ioserver 的服务 url，不带证书的使用 http，带证书的使用 https；例如本例：

<http://192.168.1.214:923>

会提示输入账号密码，默认的账号是权限最低的 operator



ioserver 一共内置了 3 个账号，默认密码参见 2.2.4 一节。

### 4.2 设备运行状态

默认打开的就是 index.html，设备运行状态在页面左侧。

上部分为对实时库的连接状态。

- Status: 每个库连接状态，online/offline
- url: 每个库当前使用的连接 url。



下半部分为设备状态表，可以单独操作启停设备。

- Status: 设备运行状态。
- HA\_status: 高可用状态, NONE 表示独立运行, 没有配置双机热备冗余, MASTER 表示主工作状态, SLAVE 表示处于热备状态。
- Tags 表示标签数。
- Ctrl 一栏是启停控制图标。点击可以启动或停止该设备。

ioserver run status

← → ↻ ⚠ 不安全 | 192.168.1.214:923/#

ioserver - 运行状态

Link	Status	Url
master_db	ONLINE	ws://192.168.1.214:921

Device	Status	HA status	Tags	Ctrl
mod1	ERROR	NONE	145	🔄
simu1	OK	NONE	1,024	🔌
ua1	ERROR	NONE	113	🔄

simu1

标签 \*

描述 \*

No.	Name	Description	Unit
60	d0.f26.pv	desc_d0.f26	u_d0.f26
61	d0.f27.pv	desc_d0.f27	u_d0.f27
62	d0.f28.pv	desc_d0.f28	u_d0.f28
63	d0.f29.pv	desc_d0.f29	u_d0.f29
64	d0.f30.pv	desc_d0.f30	u_d0.f30

点击设备名可以在右边显示该设备的标签表和当前快照数据。

功能导航 ▾

operator

simu1

标签 \*

描述 \*

类型 \*

查询

标签数:1024

每页 20 行

3 / 51 页

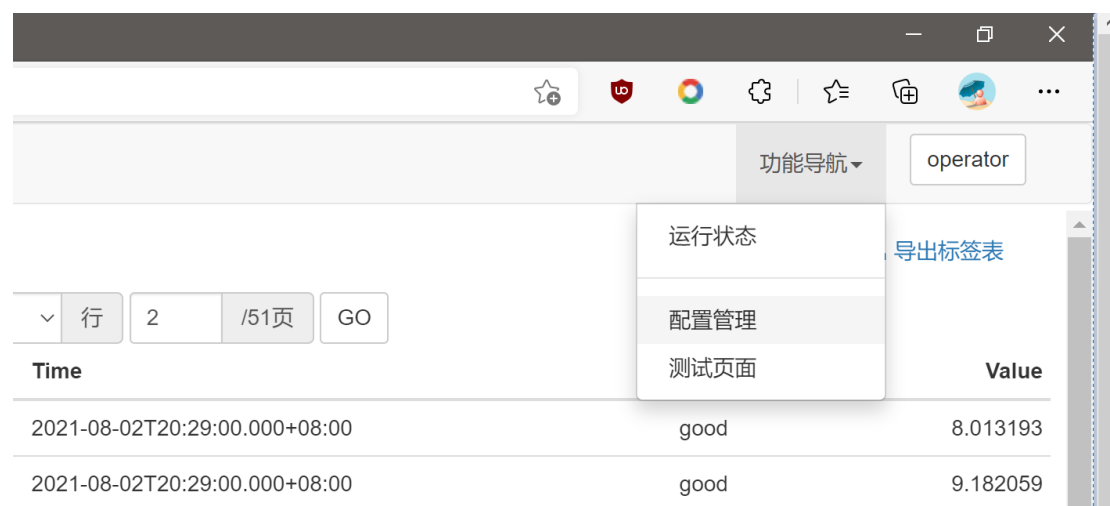
GO

No.	Name	Description	Unit	PointType	Access	Time	Quality	Value
60	d0.f26.pv	desc_d0.f26	u_d0.f26	float	R	2021-08-04T16:42:11.000+08:00	good	25.993544
61	d0.f27.pv	desc_d0.f27	u_d0.f27	float	R	2021-08-04T16:42:11.000+08:00	good	26.954834
62	d0.f28.pv	desc_d0.f28	u_d0.f28	float	R	2021-08-04T16:42:11.000+08:00	good	27.916122
63	d0.f29.pv	desc_d0.f29	u_d0.f29	float	R	2021-08-04T16:42:11.000+08:00	good	28.877413

点击导出标签表可以当前设备的标签标签表，注意只有 engineer 和 admin 才具有此权限。operator 无此权限。注意导出的是可以导入实时库的标签表，字符编码是 UTF8，使用 excel 打开修改时注意更改 excel 的编码为 utf8，否则汉字会出现乱码。

## 4.3 配置管理

在页面右上角的地方选在功能菜单的配置管理项进入配置管理页面。



左边是配置文件列表，点击文件名可以在右边显示。

ioserver configure

×

+

←

→

↻

⚠ 不安全 | 192.168.1.214:923/config.html#

A

ioserver - 配置管理

Config file	Del	Size	↻
opcua1tag.csv	🗑	5,376	☁
mod_cfg.ini	🗑	1,435	☁
mod_exp.csv	🗑	71	☁
mod_tags.csv	🗑	5,683	☁
<u>io-device.json</u>	🗑	522	☁
io-calc.csv	🗑	189	☁
io-dblink.json	🗑	152	☁
simu1.ini	🗑	17	☁
predotags.csv	🗑	95	☁
io-server.ini	🗑	1,252	☁
opcua1.conf	🗑	864	☁

io-device.json

```

1  {
2      "device": [{
3          "name": "mod1",
4          "driver": "dac_modrtu",
5          "config": "mod_cfg.ini",
6          "expfile": "mod_exp.csv",
7          "loglevel": "dbg",
8          "autoload": "yes",
9          "snaptime": "driver"
10     }],{
11         "name": "simu1",
12         "driver": "simudrv",
13         "config": "simu1.ini",
14         "expfile": "",
15         "loglevel": "dbg",
16         "autoload": "yes",
17         "snaptime": "driver"
18     }],{
19         "name": "ua1",
20         "driver": "dac_opcua",

```

列表中有删除和下载图标点击执行对应功能，需要 admin 或 engineer 账号。右上角可以上传配置文件，同名文件在上传后被覆盖。如果要添加和修改设备配置，可下载后修改或直接上传。

功能导航 ▾

operator

修改密码

mod\_cfg.ini

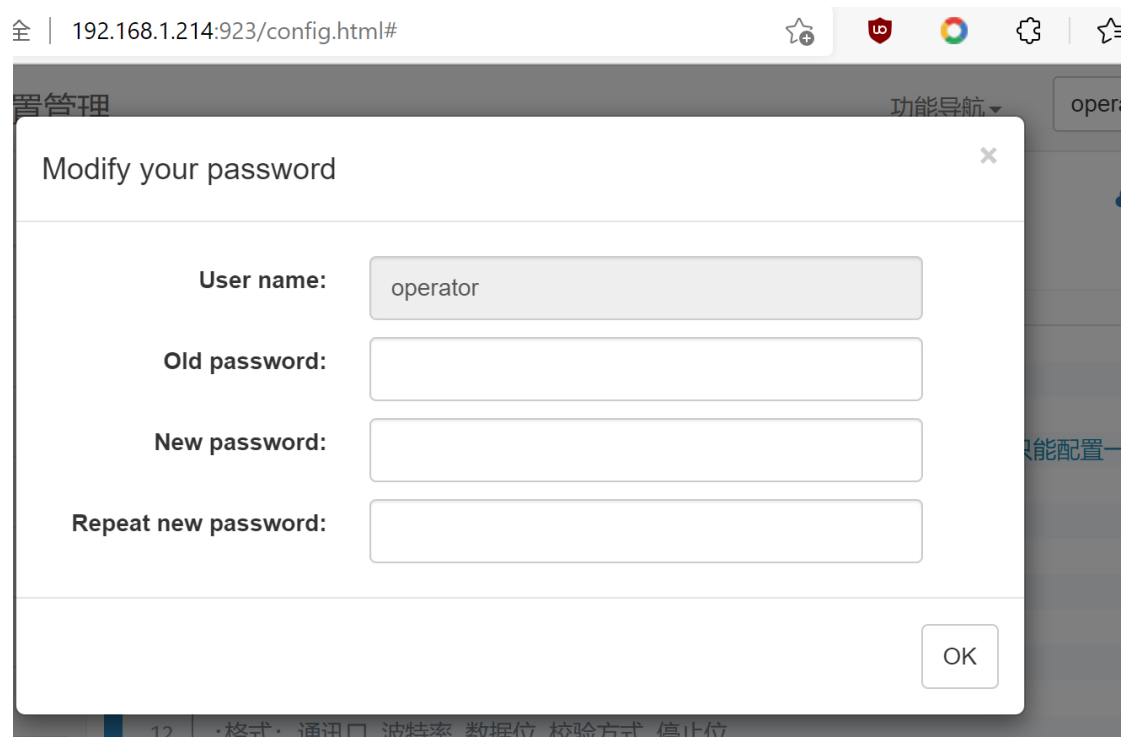
☁ 上传配置文件

```

1  ;modbus config.ini
2
3  [modbus]
4  comargs = 192.168.1.58,502 ;TCP通讯参数, 自动识别,和485参数同时只能配置一个
5  ;TCP配置 192.168.1.41,502
6  ;格式: ip地址,端口
7  ;例子: 192.168.1.41,502
8  ;-----
9

```

右上角的登录按钮显示当前登录的用户账号，点击登出，登出后显示 login 字样，在登录状态下旁边有修改当前账号密码按钮，需要输入旧密码和新密码。



The screenshot shows a web browser window with the address bar displaying "192.168.1.214:923/config.html#". The browser's address bar also shows several icons: a star, a red shield with a white 'U', a rainbow circle, a gear, and a star with a plus sign. The main content area of the browser shows a web interface with a dark header. On the left, there is a "设备管理" (Device Management) button. On the right, there is a "功能导航" (Function Navigation) dropdown menu and a "open" button. A modal dialog box titled "Modify your password" is centered on the screen. The dialog has a close button (X) in the top right corner. It contains four input fields: "User name:" with the value "operator", "Old password:", "New password:", and "Repeat new password:". An "OK" button is located at the bottom right of the dialog. The background of the web interface is partially visible, showing a table with columns for "格式" (Format), "通讯口" (Communication Port), "波特率" (Baud Rate), "数据位" (Data Bits), "校验方式" (Parity), and "停止位" (Stop Bits). The table has a row with the value "12" in the first column.

---

## 5 从 rdbdac\_ux 迁移

从 rdbdac\_ux 迁移到 ioserver 只需做少量编辑性工作，不需要做开发性工作。设备驱动程序和驱动自己配置无需改动，直接可用。仅采集值工程值转换需要重新编辑。

- 1) 安装好日志服务器 logsrv 和新网关 ioserver，将第三方驱动驱动程序拷贝到 drivers 目录，注意如果有第三方驱动的依赖库需要放在 ioserver 根目录(即和 rdbapi 相同的目录)，不要覆盖 ioserver 自带的原生驱动。
- 2) 配置实时库连接，编辑 **io-server.ini** 里的[dblink]小结，如果只有一个库直接用默认的，增加一个库直接就添加一个[dblink]小结。然后在 config 指向的文件(默认 **io-dblink.json**)里编辑实时库的连接参数，单库只需一个连接，双机热备需要配置两个连接，注意对一个高可用冗余库的热备双连接是故障是切换的，并不同时提交，数据同步由实时库通过心跳线完成。
- 3) 将设备驱动的设备配置文件拷贝到 config 目录，编辑 io-device.json 将原 rdbdac\_ux.ini 里的设备配置搬移到 io-device.json 里。
- 4) 为需要表达式转换的设备标签配置转换公式。转换配置文件是属于设备的，每个设备一个，如果设备没有转换，**io-device.json** 里配置空。
- 5) 如果有计算标签，用 ecxel 打开 **io-calc.csv** 文件添加计算标签。
- 6) 标签预处理，建议使用 fork 函数在转换表达式里一起处理，参见 3.6.1 节描述的方法。也可以用 rdbdac\_ux 的模式，将所有设备的预处理标签配置在固定文件名的 **config/predotags.csv** 里面。
- 7) 注意 ioserver 使用了新的批量控制标签注册，需要使用新的 rdb2021.7 版支持，因此还需要升级实时库 rdbsrv 或 rdbux64 到 2021.7 月版。
- 8) 如果先启动设备，然后导出标签表，将标签导入实时库后，需要重启一下该设备，以便能成功的在实时库里注册控制标签。
- 9) ioserver 不再提供越限 SOE 事件生成，请将越限报警配置到实时库服务端。对于驱动里生成的 SOE 事件同样支持提交到实时库。

## 6 常见问题

出现问题一般都可从过日志分析出故障原因。

### 6.1OPCDA 无数据

Ioserver 是后台方式运行，system 账号，加载的 IODevice 进程也是继承使用 system 账号，基于安全规范，一般的 OPCDA server 都无法使用 system 账号连接。需要改变 ioserver 服务的登录身份，直接在“服务”里修改 ioserver 属性的登录账号。如下图：



从“本地系统”账户改成使用“此账户”，一般填写当前 Windows 桌面登录的账户，如果 ioserver 中没有 opcda 设备，可以保持默认的“本地系统”账户。

如果个别标签无数据，检查该标签的标签名是不是太长，超过超过 79 字节。如果是中文标签没有数据，尝试改变 opcda 配置标签表的字符编码，一般为 ANSI 编码(本地 gbk 编码)。注意中文名转换为 utf8 后长度也不能超过 79 字节。

### 6.2OPCDA 设备启动失败

检查是不是使用的 rdb2021.8 版中的 OPCDA 驱动，因依赖库的访问规则变更，原 rdbdac\_ux\_win 中的 opcda 驱动不能用于 ioserver。检查 ezopc\_c.dll 是否在 ioserver 的根目录(即和 rdbapi.dll 相同的目录)。

---

## 6.3 设备无法启动

检查 `io-device.json` 中设备配置, 包括: 驱动程序名, 该驱动的配置文件名是否正确。

如果驱动程序有其他依赖库, 依赖库需要放在 `ioserver` 的根目录 (即和 `rdbapi.dll` 相同的目录), 如果驱动有特殊的依赖库放置说明 (比如有些 linux 下的驱动指明依赖库放在 `/usr/lib` 或 `/usr/local/lib` 目录), 则按照说明放置。

## 6.4 WEB 页面出现乱码

使用 `notepad++` 文本编辑器检查该设备自己的标签表文件的编码不能有混码, 要么是 ANSI (本地 `gbk` 编码, 中文 windows 的默认编码), 要么是 `utf-8` 编码。`Ioserver` 在 web 页面显示时, 会检查是否是 `utf8` 编码, 不是则认为是 `gbk` 编码并转换为 `utf8` 编码显示。