

RDB 数据交换协议

(RDB data exchange protocol for inver 5123)

重庆唐码软件有限公司

2025 年 4 月 update

内容目录

1. 概述	4
1.1 协议层次	4
1.2 TLS1.2 规范	5
1.3 websocket 规范	5
1.3.1 websocekt 压缩扩展	6
1.3.2 连接 url 地址	6
1.3.3 兼容性	6
1.4 JSON 规范	6
2 应用层数据交换协议 (DXP)	8
2.1 报文规范	8
2.1.1 请求/应答报文	8
2.1.2 服务器推送消息	9
2.2 登录实时库	10
2.3 时标	12
3. 数据查询统计类命令详解	14
3.1 读取快照 rdb_getsnap	14
3.2 查询值标签历史 rdb_valquery	18
3.3 读取绘图数据 rdb_plotdata	23
3.4 读取值标签历史断面值 rdb_valgetsection	28
3.5 读取对象标签历史 rdb_objget	31
3.6 值标签数据统计 rdb_sum	34
3.7 值标签运行时间统计 rdb_countruntime	37
3.8 值标签变化统计 rdb_countkst	39
3.9 读取超限标签快照 rdb_getoutlimitsnaps	41
3.10 订阅标签快照 rdb_sscsnaps	43
3.11 取消标签快照订阅 rdb_unsscsnaps	46
3.12 取消所有标签快照订阅 rdb_unsscallsnaps	47
3.13 读取标签快照订阅表 rdb_listsscsnaps	48
3.14 快照推送 rdb_pushsnaps	49
3.15 统计增量 rdb_countincrement	51
3.16 统计积分累积量 rdb_countintegral	53
4 数据写入删除类命令详解	55
4.1 写入快照 rdb_writesnaps	55
4.2 补录历史 rdb_insert	57
4.3 写设备(控制输出)rdb_writedevice/sub_writedevice	60
4.4 人工置数 rdb_manual_set/sub_manual_set	62
4.5 取消人工置数 rdb_manual_del/sub_manual_del	64
4.6 注册控制标签 rdb_regctrltag	66
4.7 控制输出 ctrl_device	67
4.8 无压缩插入历史 rdb_insertex	67
4.9 删除标签数据 rdb_datadelete	69
4.10 读取当前工作模式 rdb_getworkmode	71

4.11 设置工作模式 <code>rdb_setworkmode</code>	72
5 标签管理命令详解	74
5.1 读取标签属性 <code>rdb_tagget/sub_tagget</code>	74
5.2 查询标签 <code>rdb_tagquery/sub_tagquery</code>	81
5.3 标签的添加和修改 <code>rdb_tagimport</code>	86
5.4 删除标签 <code>rdb_tagdel</code>	92
5.5 导入标签的扩展属性 <code>rdb_tagimport_ext</code>	93
5.6 读取标签的扩展属性 <code>rdb_tagget_ext</code>	95
5.7 删除标签的扩展属性	97
5.8 导出标签表 <code>rdb_tagexport</code>	97
6 账号管理命令详解	99
6.1 添加/修改角色 <code>rdb_addactor</code>	99
6.2 读取角色 <code>rdb_listactor</code>	101
6.3 删除角色 <code>rdb_delactor</code>	103
6.4 添加/修改账号 <code>rdb_addoperator</code>	104
6.5 读取账号列表 <code>rdb_listoperator</code>	106
6.6 删除账号 <code>rdb_deloperator</code>	108
6.7 修改密码 <code>rdb_modfypswd</code>	108
7 SOE 事件命令详解	111
7.1 查询 SOE 事件 <code>rdb_soequery</code>	111
7.2 事件写入 <code>rdb_soewrite</code>	116
7.3 事件更新 <code>rdb_soeupdate</code>	118
7.4 事件订阅 <code>rdb_soessc</code>	121
7.5 事件推送 <code>put_soe</code>	122
7.6 删除 SOE 数据 <code>rdb_soedelete</code>	124
8 系统监控和维护命令详解	126
8.1 读取在线子站列表	126
8.2 读取实时库运行信息	127
8.3 读取实时库连接会话列表	129
8.4 断开实时库连接会话连接	130
8.5 重建指定标签历史数据索引	131
8.6 内存池信息查询	132
9 协议实现	138
9.1 <code>websocket</code> 的 PING/PONG 消息	138
10 更新记录	139

1.概述

RDB data exchange protocol(DXP)是实时数据库 RDB_2020.3 版开始支持的全新的数据交换协议，传输层采用标准通道(WS/WSS)，交互数据描述层采用 JSON 对象表示。支持各种客户端使用，包括移动客户端，WEB 客户端，C/S 客户端接入。

本文给出的协议，可使用 JavaScript，java，C#，C/C++实现客户端 API。RDB_2020.3 提供了一个兼容发以前版本的用 C/C++实现的 windows/Linux 版的 rdbapi，带 java jni 接口，支持 C#，C/C++，java 应用程序使用。

1.1 协议层次

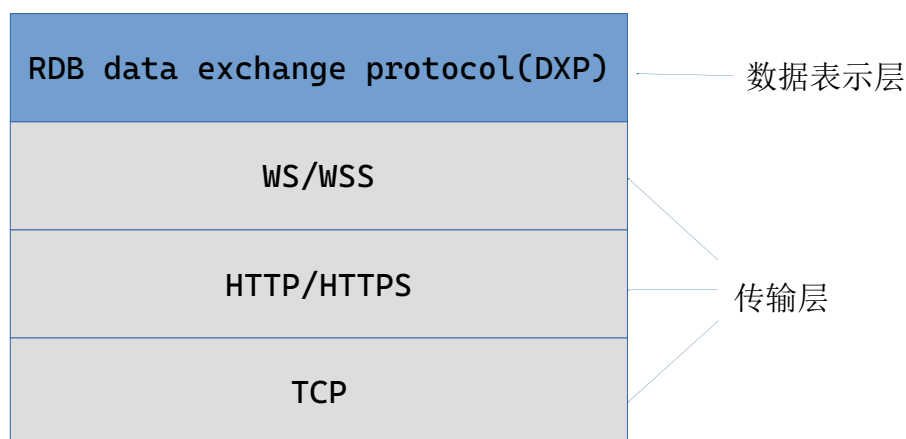


图 1 协议层次

HTTP：超文本协议。

HTTPS：这里是指基于 TLS1.2 私密通道的超文本协议

WS：从 HTTP 升级而来的 websocket 协议。

WSS：从 HTTPS 升级而来的 websocket 协议。

DXP：实时库使用 JSON 对象表示的数据交换协议。

所有以上这些协议，TOM 实时库都是原生支持，无需第三方库或者组件，支持裸系统部署，支持嵌入式 ARM Linux 平台部署。

1.2 TLS1.2 规范

安全性方面，TOM 实时库支持基于 TLS 1.2(RFC5246)版本的安全通道的 HTTPS 和 WSS 规范，具体支持的密码套件：

- TLS_RSA_WITH_AES_128_CBC_SHA256 = { 0x00,0x3C };
- TLS_RSA_WITH_AES_256_CBC_SHA256 = { 0x00,0x3D };
- TLS_RSA_WITH_AES_128_CBC_SHA = {0x00,0x2F};
- TLS_RSA_WITH_AES_256_CBC_SHA = {0x00,0x35};

即使不采用 TLS 安全私密通道，TOM 实时库的登陆验证过程也是无密码交换的防重放攻击的登录方式。参见第 2 章的登录验证。

1.3 websocket 规范

websocket 协议(RFC6455)是 html5 组成之一，是在 http 协议上的升级协议，浏览器中的 javascript 原生提供 websocket 客户端 API。使用 websocket 客户端可以实时的和服务器通信。

rdbsrv 提供了原生的 http 协议和 websocket 协议支持(不依赖任何第三方软件和第三方库,直接在 tcp 层提供协议支持)。http 协议为 1.1 版，websocket 协议为 13 版。

GET / HTTP/1.1

Sec-WebSocket-Version:13

为了便于描述，在 rdb5 中基于 websocket 的应用层协议命名为 rdbws 协议。websocket 协议支持文本和二进制模式，为方便 javascript 处理，rdbws 协议采用文本模式，服务器发送的报文采用字符串模式 JSON 对象。

其中 http 协议支持 GET 和 HEAD 命令，GET 命令除了用户升级 websocket 协议外，还可作为一个无后台脚本支持的简易 WEB 发布服务器(参见例子中 rdb_http/tom/index.html)，由于 rdbsrv 提供的 http 协议是只读的，没有 PUT、POST、DELETE 等命令，因此是绝对安全的。

http 协议的 GET 命令和 websocket 协议，不依赖其他 WEB 发布软件，只是 rdbsrv 自己就能足够支持使用客户端 javascript 脚本开发基于实时库的 WEB 应用了，比如基于 WEB 的历史报表，基于 html5 中 SVG 或 canvas 技术 + javascript 的 web 图形客户端应用。

1.3.1 websocket 压缩扩展

实时数据库的 **websocket** 接口支持压缩扩展，特别适合云服务器等网络资源非常宝贵的场合。目前支持两种压缩扩展标准：

- **permessage-deflate**: 目前是最新的标准(RFC7692), **chrome,firefox** 等都支持。
- **deflate-frame**: 已废弃的标准，目前只有 **IOS** 的 **safari** 在使用，扩展名为 **x-webkit-deflate-frame**, **safari** 并不支持 RFC7692 制定的 **permessage-deflate** 压缩扩展，为了节约移动端苹果用户的流量，实时库依旧支持这个标准。

1.3.2 连接 url 地址

WS_URL 例子：

`"ws://192.168.1.61:921"`

如果使用 80 端口则：`"ws://192.168.1.61"`

WSS_URL 例子：

`"wss://192.168.1.61:921",`

如果使用 443 端口：`"wss://192.168.1.61"`

和 HTTP 协议的 URL 中指定端口一样，如果使用标准端口(**WS:80, WSS443**)则无需指定。

1.3.3 兼容性

支持 **html5** 标准的客户端浏览器均支持，目前已测试兼容的客户端浏览器包括：

- **MicroSoft Edge for windows**
- **Chrome for windows/Linux**
- **Chromium for Linux**
- **Firefox for windows/Linux**
- **Safari for IOS 9, 10, 11 ipad and iphone**
- **Android 4.0 及以上手机平板内置浏览器**

1.4 JSON 规范

JSON 规范使用 **RFC8259** 标准。需要特别提示的是：

- 字符串均为 **utf8** 编码的 **unicode** 字符串。
- 使用双引号分离字段名和字符串类型的字段值，字段值中出现双引号需要使用字符 `'\"'` 转义。
- 不能有注释。`//`和`/**/`都不能有(规范也没有)。
- **2023.7(内部版本 5108)**之前除了支持双引号的转义，其他转义不支持，比如 `'\b'`, `'\t'`, `"\u002F"` 这些都不支持。从 **2023.7(内部版本 5108)**开始完全支持 **RFC8259** 标准约定的所有转义。

Bray

Standards Track

[Page 8]

FF

RFC 8259

JSON

December 2017

To escape an extended character that is not in the Basic Multilingual Plane, the character is represented as a 12-character sequence, encoding the UTF-16 surrogate pair. So, for example, a string containing only the G clef character (U+1D11E) may be represented as `"\uD834\uDD1E"`.

```
string = quotation-mark *char quotation-mark

char = unescaped /
      escape (
        %x22 /      ; "    quotation mark  U+0022
        %x5C /      ; \    reverse solidus  U+005C
        %x2F /      ; /    solidus          U+002F
        %x62 /      ; b    backspace        U+0008
        %x66 /      ; f    form feed        U+000C
        %x6E /      ; n    line feed        U+000A
        %x72 /      ; r    carriage return  U+000D
        %x74 /      ; t    tab              U+0009
        %x75 4HEXDIG ) ; uXXXX              U+XXXX

escape = %x5C          ; \

quotation-mark = %x22  ; "

unescaped = %x20-21 / %x23-5B / %x5D-10FFFF
```

2 应用层数据交换协议 (DXP)

客户端和服务端使用 JSON 对象来交互信息，通过 WS/WSS 通道的文本模式向对端发送。交互方式分为请求/应答和服务端主动推送两种，采用异步模式，无需等待命令处理完成。

2.1 报文规范

2.1.1 请求/应答报文

每个请求报文必有两个字段：**request** 和 **seqno**；

request 是请求命令，字符串型。

seqno 是大于 0 的整数，客户端填写，服务端原样返回，用于 **request/response** 配对使用，建议每次请求 **seqno** 加 1，达到 MAX_INT32(2,147,483,647)后重新从 1 开始。

例如：

```
{
  "request": "rdb_getsnap",
  "seqno": 2002,
  "tags": [
    "sub1:opc1.r_f32",
    "sub1:opc1.r_f64"
  ]
}
```

每个应答报文必有 3 个字段：**response**, **seqno**, **status**

response 字符串型，填写的客户端的 **request** 字段值。

seqno 整数，原样填写客户端请求报文中的值。

status 整数，处理结果状态，0 表示成功，其他为错误码。

例如：

```
{
  "response": "rdb_getsnap",
  "seqno": 2002,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "N": "sub1:opc1.r_f32",
      "E": 0,

```



```

        "DT": 3,
        "T": "2020-03-01 17:50:57.000",
        "Q": 1,
        "V": 0.6257
    },
    {
        "N": "sub1:opc1.r_f64",
        "E": 0,
        "DT": 5,
        "T": "2020-03-01 17:50:57.000",
        "Q": 1,
        "V": 56.257
    }
]
}

```

2.1.2 服务器推送消息

当客户端订阅 SOE 事件后，如果服务端有符合订阅条件的事件会主动推送给客户端。

消息必有 **response**，**seqno** 字段。

response 字符串型，表明推送的是什么内容，目前这里只有一个值 **"put_soe"**。

seqno 整数，其中 **seqno** 为 0 表示是服务器主动推送，所以客户端的请求报文不要使用 0。

例如 **soe** 推送报文：

```

{
    "response": "put_soe",
    "seqno": 0,
    "status": 0,
    "message": "OK",
    "vals": [
        {
            "time": "2020-03-11 10:08:20.000",
            "ukey": 0,
            "type": 0,
            "argtype": -1,
            "arglen": 27,
            "level": 0,
            "source": "rdbapidemo",
            "des": "",

```

```

        "sarg": "No1:2020- 3-11 10: 8:20. 0",
        "cflag": 0
    },
    {
        "time": "2020-03-11 10:08:20.000",
        "ukey": 1,
        "type": 0,
        "argtype": 101,
        "arglen": 8,
        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "yNASsAMAAAA=",
        "cflag": 0
    }
]
}

```

2.2 登录实时库

登录采用无密钥交换的两次握手方式，即使不走 **WSS** 私密通道也无法截获密钥。

其原理为：客户端发送登录报文，带有自己的用户名(账号)，服务端回答一个随机字符串。客户端使用随机字符串加上自己密码的 **MD5** 字符串在做 **MD5** 散列计算，并将结果发给服务端，服务端用同样方法验证客户端是否合法。

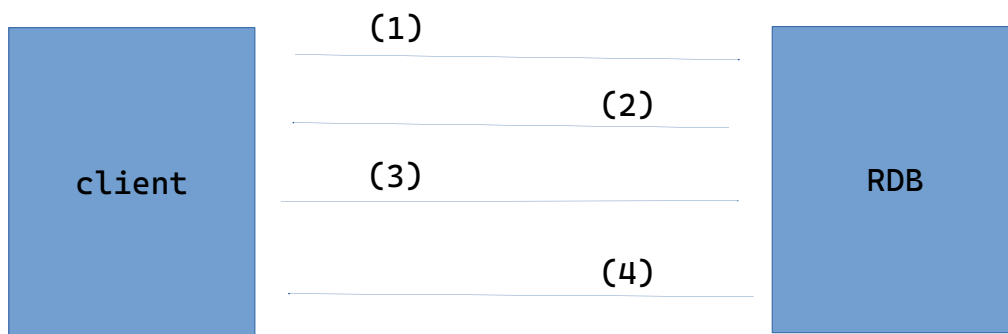


图 2 登录交互图

(1)client 发送 rdb_login 报文

```

{
    "request": "rdb_login",
    "seqno": 1,
    "name": "admin",

```

```
    "timefmt": "iso"
}
```

其中 **name** 字段为 **client** 的用户名，这里使用 **admin** 作为例子。

timefmt 字段为可选字段，协商服务器返回的时标格式，自 **rdb2023.3**（内部版本 **5104**）开始支持。**timefmt** 字段可以为 **"iso"**、**"jsnum"**、**"local"** 之一（具体格式见 **2.3** 时标），不提供此字段表示使用原 **local** 格式。这里协商的 **timefmt** 只是要求服务器返回数据的时标格式，对于客户端提交给服务器的数据时标可以任意格式甚至不同记录采用不同时标格式。

协商时标目的时兼容以前版本的客户端，以前版本的客户端登录时不会提供 **timefmt** 字段，服务器会按照原格式（服务器本地时标）返回数据。

从 **2023.4**（内部版本 **1015**）开始，可以参考 **4.11** 描述的提供 **workmode** 字段指定登录后的工作模式。

(2) RDB 应答 **rdb_login** 报文

```
{
    "response": "rdb_login",
    "seqno": 1,
    "version": 5104,
    "status": 0,
    "vals": "4F2BB96A64DE184DE4CA65765645B55D",
    "timefmt": "iso"
}
```

其中 **vals** 为服务端发送的随机字符串。

自 **2023.3** 版开始，会返回 **version**（服务器内部版本）和 **timefmt** 字段。

如果有 **timefmt** 字段返回，表示登录成功后服务端发送给客户端的数据中的时标时标会按照此字段指示的格式处理。如果服务器没有返回这个字段（**2023.3** 版之前的 **rdbsrv** 不会返回此字段），客户端提交的数据需要按照原时标模式（服务器本地时标）处理。

如果失败，则会返回无此用户。例如：

```
{
    "response": "rdb_login",
    "seqno": 1,
    "status": 33,
    "message": "no user!"
}
```

(3)client 使用随机数和自己的密码做 MD5 散列计算

这里假设密码也是"admin", 客户端在收到的 vals 字段的字符串的尾部加上自己密码的 MD5 的 32 字节 HEX 大写书写格式的字符串

("21232F297A57A5A743894A0E4A801FC3")组成新的字符串

(4F2BB96A64DE184DE4CA65765645B55D21232F297A57A5A743894A0E4A801FC3),

然后再计算新字符串的 MD5 摘要转换为 32 字节的 HEX 大写书写格式

("E1446FC255396A2AF49C0F8F6F5F8466")使用 rdb_auth 命令发送给 RDB:

```
{
    "request": "rdb_auth",
    "seqno": 2,
    "vals": "E1446FC255396A2AF49C0F8F6F5F8466"
}
```

(4)RDB 应答验证

服务端会使用相同的算法计算验证字符串, 如何匹配表示客户端合法, 则会返回成功报文, 如下:

```
{
    "response": "rdb_auth",
    "seqno": 2,
    "version": 5104,
    "status": 0,
    "vals": "login success",
    "timefmt": "iso"
}
```

验证成功, 自 2023.3 版开始, 同样会返回 version 和 timefmt 字段。

如果验证失败会返回密码错误报文, 如下:

```
{
    "response": "rdb_auth",
    "seqno": 2,
    "status": 34,
    "message": "password error!"
}
```

2.3 时标

在以前版本中, 协议中时标字段均是服务器本地时标, 从 2023.3 版(内部版本 5104)开始支持带时区格式的时标(IS08601 格式)和 JavaScript timestamp 类型的数

字型时标(GMT 毫秒数, 即 0 时区自 1970-1-1 开始的毫秒数, JSON 中为 **number** 类型, 与时区无关)。

- 1) **rdbsrv** 接收数据中的时标, 包括命令参数中时标, 自 2023.3 版开始自动识别三种格式:

local(服务器本地时标): 例如 "2023/2/20 12:10:32.123"

iso(ISO8601 时标): 例如 "2023-2-20T12:10:32.123+08:00"

Jsnum(JavaScript 数据格式, UTC1970-1-1 开始的毫秒数) 例如
1676866232123

以上三个时标都是一样的。

- 2) **rdbsrv** 返回的数据中的时标, 会按照登录协商的格式, 如果没有协商, 按照原有 **local** 格式(例如 "2023/2/20 12:10:32.123"), 登录时的时标协商规范参见 2.2 登录实时库。

3.数据查询统计类命令详解

本章描述的命令是读取查询类命令集，包括实时数据，历史数据，统计，分别从主站和子站访问。

3.1 读取快照 rdb_getsnap

rdb_getsnap 从主站读取快照，**sub_getsnap** 通过主站代理读取指定子站的快照。读取值标签和对象标签都是一样。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_getsnap"读取主站标签快照 "sub_getsnap"读取子站标签快照
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_getsnap"时有效
tags	string array	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_f32" 和 "opc1.r_f32" 等价

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_getsnap"读取主站标签快照 "sub_getsnap"读取子站标签快照
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_getsnap"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。

vals	object array	记录集，JSON 对象数组，每个对象字段定义： "N" : string; 标签名; "E" : int ; 错误码; 0 表示成功，其他为错误码 "DT": int ;数据类型, DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7; "Q" : int; 数据质量; 无此字段或者 0 表示 good "T" : string 或者 number, 参见 2.3 时标 "V" : 标签值; 其中 object 对象标签使用 base64 编码
------	--------------	---

例子:

请求 1(读取主站快照):

```
{
  "request": "rdb_getsnap",
  "seqno": 2002,
  "tags": [
    "opc1.r_f32",
    "opc1.r_str",
    "opc1.r_blob"
  ]
}
```

应答 1:

```
{
  "response": "rdb_getsnap",
  "seqno": 2002,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "N": "opc1.r_f32",
      "E": 0,
      "DT": 3,
      "T": "2020-03-01 17:50:57.000",
      "Q": 1,
      "V": 0.6257
    }
  ]
}
```

```

    },
    {
        "N": "opc1.r_str",
        "E": 0,
        "DT": 6,
        "T": "2020-03-01 17:50:54.000",
        "Q": 0,
        "V": "2020-03-01 17:50:54"
    },
    {
        "N": "opc1.r_blob",
        "E": 0,
        "DT": 7,
        "T": "2020-03-01 17:50:54.000",
        "Q": 0,
        "V": "MjAyMC0wMy0wMSAxNzo1MDo1NA=="
    }
]
}

```

请求 2，读取子站快照(标签名不带子站信息):

```

{
    "request": "sub_getsnap",
    "seqno": 2003,
    "station": "sub1",
    "tags": [
        "opc1.r_f32",
        "opc1.r_str",
        "opc1.r_blob"
    ]
}

```

应答 2:

```

{
    "response": "sub_getsnap",
    "station": "sub1",
    "seqno": 2003,
    "status": 0,
    "message": "OK",
    "vals": [
        {
            "N": "opc1.r_f32",

```



```

        "E": 0,
        "DT": 3,
        "T": "2020-03-11 17:07:34.000",
        "Q": 1,
        "V": 0.7654
    },
    {
        "N": "opc1.r_str",
        "E": 0,
        "DT": 6,
        "T": "2020-03-11 17:07:32.000",
        "Q": 0,
        "V": "2020-03-11 17:07:32"
    },
    {
        "N": "opc1.r_blob",
        "E": 0,
        "DT": 7,
        "T": "2020-03-11 17:07:32.000",
        "Q": 0,
        "V": "MjAyMC0wMy0xMSAxNzowNzozMg=="
    }
]
}

```

请求 3，读取子站快照(标签名带子站信息)

```

{
    "request": "sub_getsnap",
    "seqno": 2004,
    "station": "sub1",
    "tags": [
        "sub1:opc1.r_f32",
        "sub1:opc1.r_str",
        "sub1:opc1.r_blob"
    ]
}

```

应答 3:

```

{
    "response": "sub_getsnap",
    "station": "sub1",
    "seqno": 2004,

```

```

"status": 0,
"message": "OK",
"vals": [
  {
    "N": "sub1:opc1.r_f32",
    "E": 0,
    "DT": 3,
    "T": "2020-03-11 17:07:34.000",
    "Q": 1,
    "V": 0.7654
  },
  {
    "N": "sub1:opc1.r_str",
    "E": 0,
    "DT": 6,
    "T": "2020-03-11 17:07:32.000",
    "Q": 0,
    "V": "2020-03-11 17:07:32"
  },
  {
    "N": "sub1:opc1.r_blob",
    "E": 0,
    "DT": 7,
    "T": "2020-03-11 17:07:32.000",
    "Q": 0,
    "V": "MjAyMC0wMy0xMSAxNzowNzozMg=="
  }
]
}

```

3.2 查询值标签历史 rdb_valquery

此命令包含读取主站，子站的样本值，插值，绘图数据，区间最大最小值。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_valquery"从主站查询 "sub_valquery"从子站查询。
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_valquery"时有效

tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_f32" 和 "opc1.r_f32" 等价
exp	string	可选，查询表达式；从内部版本 5114 开始，当 flag=1,3 读取绘图数据时忽略 exp 参数。
time_begin	String /Number	string 或者 number, 参见 2.3 时标
time_end	String /Number	可选，string 或者 number, 参见 2.3 时标，不提供此字段表示没有结束时间一直读完；如果时读取绘图数据，不提供 time_end 且 dt=0 表示 time_end 为从 time_begin 开始后的一天并不超过当前时间。
dt	int	可选，不填默认值为 0。 0 表示样本值；从内部版本 5114 开始，当 flag=1,3 读取绘图数据时，dt 为 0 表示服务器通过 time_begin、time_end 和 top 值计算间隔。 >0 表示间隔的 100 毫秒数(比如 600 表示间隔一分钟)
flag	int	可选，读取方式，不填表示 0 模式 0：默认模式，QA = 1 的 shutdown 的时间段数据不插值。 1：绘图数据，插值包含样本值，因此插值出来的记录数据多余预计的记录数，QA = 1 的 shutdown 的时间段数据不插值 2：全数据插值，不管数据质量 3：全数据绘图数据，不管数据质量，插值包含样本值，因此插值出来的记录数据多余预计的记录数 4：读取区间最小值，内部版本 5114 及以后支持。 5：读取区间最大值，内部版本 5114 及以后支持。 6：读取区间最小值，不管数据质量，内部版本 5114 及以后支持。 7：读取区间最大值，不管数据质量，内部版本 5114 及以后支持。 8：冷备份模式，只读取归档数据，用于冷备实时库，和源库归档记录完全一致，此时 lds 和 sexp 参数不起作用。内部版本 5122 及以后支持。与之配套的 xdbsyn 冷备工具采用就是此模式。 注 1)：4，5，6，7 模式 dt 必须 > 0 表示区间(等于 0 会返回样本值)，表达式 exp 参数被忽略。当区间内无样本值记录时，会采用插值方式补齐，插值模式根据标签属性中的定义决定使用梯形还是线性插值。4，5 模式 QA = 1 的 shutdown 的时间段数据不插值；6，7 模式则不关心 QA 的值。 注 2) 1，3 模式绘图数据从内部版本 5114 开始，表达式 exp 参数被忽略；dt 为 0 表示服务器通过

		time_begin 、 time_end 和 top 值计算间隔：不提供 time_end 表示从 time_begin 开始后的一天并不超过当前时间。
top	int	最多读取多少个 dt 时间段数据，可选，默认 100 ，最大 2048 （绘图数据最大 1280 ）。

应答字段说明

KEY	VAL 类型	说明
response	string	" rdb_valquery "读取主站标签历史 " sub_valquery "读取子站标签历史
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名， request ="sub_valquery"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
end	int	0:未完； 1: 结束，时间段内没有再多的记录
vals	object array	记录集，JSON 对象数组，每个对象字段定义： "N" : string ； 标签名； "E" : int ； 错误码； 0 表示成功，其他为错误码 "DT" : int ；数据类型， DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7; "Q" : int ； 数据质量； 0 表示 good，无此字段也表示 good "T" : string 或 number ， 参见 2.3 时标描述 "V" : 标签值；

例子:

请求 1 从主站查询

```
{
  "request": "rdb_valquery",
  "seqno": 2009,
  "tag": "opc1.r_f32",
  "exp": "val > 0",
  "time_begin": "2020/3/1",
  "time_end": "2020/3/2",
```

```
    "dt": "0",  
    "flag": 1,  
    "top": 4  
}
```

应答 1

```
{  
  "response": "rdb_valquery",  
  "seqno": 2009,  
  "status": 0,  
  "message": "OK",  
  "end": 0,  
  "vals": [  
    {  
      "T": "2020-03-01 11:15:00.000",  
      "Q": 0,  
      "DT": 3,  
      "V": 0.25  
    },  
    {  
      "T": "2020-03-01 11:32:16.000",  
      "Q": 1,  
      "DT": 3,  
      "V": 0.3536  
    },  
    {  
      "T": "2020-03-01 17:11:02.100",  
      "Q": 0,  
      "DT": 3,  
      "V": 0.3862  
    },  
    {  
      "T": "2020-03-01 17:50:57.000",  
      "Q": 1,  
      "DT": 3,  
      "V": 0.6257  
    }  
  ]  
}
```

请求 2 从子站查询

```

{
  "request": "sub_valquery",
  "seqno": 2010,
  "station": "sub1",
  "tag": "sub1:opc1.r_f32",
  "exp": "val > 0",
  "ts": "2020/3/1",
  "te": "2020/3/2",
  "dt": "0",
  "flag": 1,
  "maxrecs": 4
}

```

应答 2

```

{
  "response": "sub_valquery",
  "station": "sub1",
  "seqno": 2010,
  "status": 0,
  "message": "OK",
  "end": 0,
  "vals": [
    {
      "T": "2020-03-01 11:15:00.000",
      "Q": 0,
      "DT": 3,
      "V": 0.25
    },
    {
      "T": "2020-03-01 11:32:16.000",
      "Q": 1,
      "DT": 3,
      "V": 0.3536
    },
    {
      "T": "2020-03-01 17:11:02.100",
      "Q": 0,
      "DT": 3,
      "V": 0.3862
    },
    {
      "T": "2020-03-01 17:50:57.000",

```

```

        "Q": 1,
        "DT": 3,
        "V": 0.6257
    }
}
]
}

```

3.3 读取绘图数据 rdb_plotdata

此命令包含读取主站，子站绘图数据，可以同时读取多个标签。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_plotdata"从主站查询 "sub_plotdata"从子站查询。
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_plotdata"时有效
tags	string array	标签名数组，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_f32" 和 "opc1.r_f32" 等价
time_begin	string	开始日期时间，string 或者 number,参见 2.3 时标
time_end	string	结束日期时间(可选)，string 或者 number,参见 2.3 时标，time_end 后 dt 参数将被忽略。内部版本 5114 开始支持。
dt	int	间隔时间(可选)，单位 100 毫秒数，600 表示 1 分钟。提供 time_end 后，dt 参数将被忽略。
flag	int	可选，读取方式，不填表示 1 模式 1：默认模式，绘图数据，插值包含样本值，因此插值出来的记录数据多余预计的记录数，QA = 1 的 shutdown 的时间段数据不插值 3：全数据绘图数据，不管数据质量，插值包含样本值，因此插值出来的记录数据多余预计的记录数
top	int	可选，最多读取多少 dt 时间段的数据，最大 1280；如果仅提供开始结束时间，不提供 dt 和 top，则由服务器决定将开始结束时间分为 1280 段读取绘图数据。

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_plotdata"读取主站标签绘图数据

		"sub_plotdata"读取子站标签绘图数据
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_plotdata"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
end	int	0:未完; 1: 结束, 时间段内没有再多的记录
vals	object array	<p>记录集, JSON 对象数组, 每个对象字段定义:</p> <p>tag: string 标签名;</p> <p>rec: object array 每个标签的记录集</p> <p>每个 rec 里的对象有四个字段, T,DT,Q,V</p> <p>"T" : string 或者 number, 参见 2.3 时标。</p> <p>"DT": int ;数据类型,</p> <p>DIGITAL=1;</p> <p>INT32=2;</p> <p>FLOAT=3;</p> <p>INT64=4;</p> <p>DOUBLE=5;</p> <p>STRING=6;</p> <p>OBJECT=7;</p> <p>"Q" : int; 数据质量; 0 表示 good, 无此字段也表示 good</p> <p>"V" : 标签值;</p>

例子:

请求 1 从主站读

```
{
  "request": "rdb_plotdata",
  "seqno": 2042,
  "time_begin": "2020-3-9 9:36:0",
  "dt": 600,
  "top": 8,
  "tags": [
    "opc1.r_i32",
    "opc1.r_f32"
  ]
}
```

应答 1


```

{
  "response": "rdb_plotdata",
  "seqno": 2042,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "tag": "opc1.r_i32",
      "rec": [
        {
          "T": "2020-03-09 09:36:00.000",
          "Q": 0,
          "DT": 2,
          "V": 7769
        },
        {
          "T": "2020-03-09 09:36:10.000",
          "Q": 0,
          "DT": 2,
          "V": 7770
        },
        {
          "T": "2020-03-09 09:43:00.000",
          "Q": 0,
          "DT": 2,
          "V": 6607
        }
      ]
    },
    {
      "tag": "opc1.r_f32",
      "rec": [
        {
          "T": "2020-03-09 09:36:00.000",
          "Q": 0,
          "DT": 3,
          "V": 0.776936
        },
        {
          "T": "2020-03-09 09:36:10.000",
          "Q": 0,
          "DT": 3,

```

```

        "V": 0.777
      },
      {
        "T": "2020-03-09 09:43:00.000",
        "Q": 0,
        "DT": 3,
        "V": 0.660791
      }
    ]
  }
]
}

```

请求 2 从子站读取

```

{
  "request": "sub_plotdata",
  "seqno": 2043,
  "station": "sub1",
  "time_begin": "2020-3-9 9:36:0",
  "dt": 600,
  "top": 8,
  "tags": [
    "sub1:opc1.r_i32",
    "sub1:opc1.r_f32"
  ]
}

```

应答 2

```

{
  "response": "sub_plotdata",
  "station": "sub1",
  "seqno": 2043,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "tag": "sub1:opc1.r_i32",
      "rec": [
        {
          "T": "2020-03-09 09:36:00.000",
          "Q": 0,
          "DT": 2,

```

```

        "V": 7769
    },
    {
        "T": "2020-03-09 09:36:10.000",
        "Q": 0,
        "DT": 2,
        "V": 7770
    },
    {
        "T": "2020-03-09 09:43:00.000",
        "Q": 0,
        "DT": 2,
        "V": 6607
    }
]
},
{
    "tag": "sub1:opc1.r_f32",
    "rec": [
        {
            "T": "2020-03-09 09:36:00.000",
            "Q": 0,
            "DT": 3,
            "V": 0.776936
        },
        {
            "T": "2020-03-09 09:36:10.000",
            "Q": 0,
            "DT": 3,
            "V": 0.777
        },
        {
            "T": "2020-03-09 09:43:00.000",
            "Q": 0,
            "DT": 3,
            "V": 0.660791
        }
    ]
}
]
}
}

```

3.4 读取值标签历史断面值 rdb_valgetsection

断面就是同一时间标签集的历史记录。

rdb_valgetsection 从主站读取，**sub_valgetsection** 从子站读取。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_valgetsection"读取主站标签断面值 "sub_valgetsection"读取子站标签断面值
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_valgetsection"时有效
time	string	string 或 number , 参见 2.3 时标描述
flag	int	可选, 读取方式, 默认值为 0 -1: 小于等于 time 0 : 等于 time , 该时间点没有则使用插值。 1 : 大于等于 time
tags	string array	标签名, 对于子站可以前面加子站名用冒号分开, 应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_f32" 和 "opc1.r_f32" 等价

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_valgetsection"读取主站标签断面值 "sub_valgetsection"读取子站标签断面值
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_valgetsection"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。

vals	object array	记录集, JSON 对象数组, 每个对象字段定义: "N" : string; 标签名; "E" : int ; 错误码; 0 表示成功, 其他为错误码 "DT": int ;数据类型, DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7; "Q" : int; 数据质量; 无此字段或者 0 表示 good "T" : string 或 number, 参见 2.3 时标描述
------	--------------	--

例子:

请求 1 读取主站

```
{
  "request": "rdb_valgetsection",
  "seqno": 2011,
  "time": "2020/3/1 17:50:57",
  "flag": 0,
  "tags": [
    "opc1.r_f32",
    "opc1.r_i32"
  ]
}
```

应答 1

```
{
  "response": "rdb_valgetsection",
  "seqno": 2011,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "N": "opc1.r_f32",
      "E": 0,
      "DT": 3,
      "T": "2020-03-01 17:50:57.000",
      "Q": 1,
      "V": 0.6257
    },
  ],
}
```

```

        {
            "N": "opc1.r_i32",
            "E": 0,
            "DT": 2,
            "T": "2020-03-01 17:50:57.000",
            "Q": 1,
            "V": 6257
        }
    ]
}

```

请求 2 读取子站

```

{
    "request": "sub_valgetsection",
    "seqno": 2012,
    "station": "sub1",
    "time": "2020/3/1 17:50:57",
    "flag": 0,
    "tags": [
        "sub1:opc1.r_f32",
        "opc1.r_i32"
    ]
}

```

应答 2

```

{
    "response": "sub_valgetsection",
    "station": "sub1",
    "seqno": 2012,
    "status": 0,
    "message": "OK",
    "vals": [
        {
            "N": "sub1:opc1.r_f32",
            "E": 0,
            "DT": 3,
            "T": "2020-03-01 17:50:57.000",
            "Q": 1,
            "V": 0.6257
        },
        {
            "N": "opc1.r_i32",

```

```

        "E": 0,
        "DT": 2,
        "T": "2020-03-01 17:50:57.000",
        "Q": 1,
        "V": 6257
    }
}
]
}

```

3.5 读取对象标签历史 rdb_objget

rdb_objget 读取主站，sub_objget 读取子站。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_objget"从主站读取 "sub_objget"从子站读取
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_objget"时有效
tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	string	开始日期时间，string 或 number，参见 2.3 时标描述
time_end	string	结束日期时间，string 或 number，参见 2.3 时标描述，不提供此字段表示没有结束时间一直读完
top	int	可选，最多读取多少条记录，默认 100，最大 256

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_objget"从主站读取 "sub_objget"从子站读取
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_objget"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
end	int	0:未完; 1: 结束，时间段内没有再多的记录

vals	object array	记录集，JSON 对象数组，每个对象字段定义： "N" : string; 标签名; "E" : int ; 错误码; 0 表示成功，其他为错误码 "DT": int ;数据类型, STRING=6; OBJECT=7; "Q" : int; 数据质量; 无此字段或者 0 表示 good "T" : string 或 number, 参见 2.3 时标描述 "V" : 标签值; string 类型为 utf8 字符串, object 类型为 base64 编码的字符串。
------	--------------	---

例子:

请求 1 从主站读取

```
{
  "request": "rdb_objget",
  "seqno": 2013,
  "time_begin": "2020/2/15 15:30:0",
  "time_end": "2020/2/15 16:0:0",
  "tag": "opc1.r_blob",
  "top": 4
}
```

应答 1

```
{
  "response": "rdb_objget",
  "seqno": 2013,
  "status": 0,
  "message": "OK",
  "end": 0,
  "vals": [
    {
      "DT": 7,
      "T": "2020-02-15 15:30:00.000",
      "Q": 0,
      "V": "MjAyMC0wMi0xNSAxNTozMDowMA=="
    },
    {
      "DT": 7,
      "T": "2020-02-15 15:30:01.000",
      "Q": 0,
      "V": "MjAyMC0wMi0xNSAxNTozMDowMQ=="
    }
  ]
}
```



```

    },
    {
        "DT": 7,
        "T": "2020-02-15 15:30:02.000",
        "Q": 0,
        "V": "MjAyMC0wMi0xNSAxNTozMDowMg=="
    },
    {
        "DT": 7,
        "T": "2020-02-15 15:30:03.000",
        "Q": 0,
        "V": "MjAyMC0wMi0xNSAxNTozMDowMw=="
    }
]
}

```

请求 2 从子站读取

```

{
    "request": "sub_objget",
    "seqno": 2013,
    "station": "sub1",
    "time_begin": "2020/2/15 15:30:0",
    "time_end": "2020/2/15 16:0:0",
    "tag": "sub1:opc1.r_str",
    "top": 4
}

```

应答 2

```

{
    "response": "sub_objget",
    "station": "sub1",
    "seqno": 2013,
    "status": 0,
    "message": "OK",
    "end": 0,
    "vals": [
        {
            "DT": 6,
            "T": "2020-02-15 15:30:00.000",
            "Q": 0,
            "V": "2020-02-15 15:30:00"
        },
    ]
}

```

```

{
    "DT": 6,
    "T": "2020-02-15 15:30:01.000",
    "Q": 0,
    "V": "2020-02-15 15:30:01"
},
{
    "DT": 6,
    "T": "2020-02-15 15:30:02.000",
    "Q": 0,
    "V": "2020-02-15 15:30:02"
},
{
    "DT": 6,
    "T": "2020-02-15 15:30:03.000",
    "Q": 0,
    "V": "2020-02-15 15:30:03"
}
]
}

```

3.6 值标签数据统计 rdb_sum

统计最大值，最小值，平均值，算数累计值。积分累计值统计请参与 3.16

rdb_sum 读取主站，**sub_sum** 读取子站。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_sum"从主站统计 "sub_sum"从子站统计。
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名， request ="sub_sum"时有效
tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	string	开始日期时间， string 或 number ，参见 2.3 时标描述
time_end	string	结束日期时间， string 或 number ，参见 2.3 时标描述，不提供此字段表示没有结束时间一直读完
exp	string	可选，查询表达式

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_sum"从主站统计 "sub_sum"从子站读取
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_sum"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
min	object	最小值产生时的记录。
max	object	最大值产生时的记录。
avg	double	平均值
sum	double	算术累计值
recordcounts	int	参与统计的样本值记录数

例子:

请求 1 主站:

```
{
  "request": "rdb_sum",
  "seqno": 2015,
  "time_begin": "2020/2/1",
  "time_end": "2020/3/1",
  "tag": "opc1.r_f32"
}
```

应答 1

```
{
  "response": "rdb_sum",
  "seqno": 2015,
  "status": 0,
  "message": "OK",
  "min": {
    "T": "2020-02-15 12:13:20.000",
    "Q": 0,
    "DT": 3,
    "V": 0
  },
}
```

```

    "max": {
      "T": "2020-02-15 12:13:19.000",
      "Q": 0,
      "DT": 3,
      "V": 0.9999
    },
    "avg": 0.370792,
    "sum": 4.8203,
    "recordcounts": 13
  }

```

请求 2 子站

```

{
  "request": "sub_sum",
  "seqno": 2016,
  "station": "sub1",
  "time_begin": "2020/2/1",
  "time_end": "2020/3/1",
  "tag": "sub1:opc1.r_f32",
  "exp": "val > 0"
}

```

应答 2

```

{
  "response": "sub_sum",
  "station": "sub1",
  "seqno": 2016,
  "status": 0,
  "message": "OK",
  "min": {
    "T": "2020-02-15 15:01:35.000",
    "Q": 0,
    "DT": 3,
    "V": 0.0095
  },
  "max": {
    "T": "2020-02-15 12:13:19.000",
    "Q": 0,
    "DT": 3,
    "V": 0.9999
  },
  "avg": 0.438209,

```

```

    "sum": 4.8203,
    "recordcounts": 11
}

```

3.7 值标签运行时间统计 rdb_countruntime

统计一个值在某一条件上的累计时间，长于用统计设备正常时间，开关正常闭合时间等。

rdb_countruntime 从主站统计，**sub_countruntime** 从子站统计。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_countruntime"从主站统计 "sub_countruntime"从子站统计。
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_countruntime"时有效
tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	String/number	开始日期时间，string 或者 number,参见 2.3 时标
time_end	String/number	结束日期时间，string 或者 number,参见 2.3 时标，不提供此字段表示没有结束时间一直读完
exp	string	可选，查询表达式

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_countruntime"从主站统计 "sub_countruntime"从子站读取
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_countruntime"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
runtimes	int64	运行时间，单位 100 毫秒数。
recordcounts	int	参与统计的样本值记录数

例子:

请求 1 主站

```
{
  "request": "rdb_countruntime",
  "seqno": 2017,
  "time_begin": "2020/2/1",
  "time_end": "2020/3/1",
  "tag": "opc1.r_i32"
}
```

应答 1

```
{
  "response": "rdb_countruntime",
  "seqno": 2017,
  "status": 0,
  "runtimes": 12526320,
  "recordcounts": 12,
  "message": "OK"
}
```

请求 2 子站

```
{
  "request": "sub_countruntime",
  "seqno": 2018,
  "station": "sub1",
  "time_begin": "2020/2/1",
  "time_end": "2020/3/1",
  "tag": "sub1:opc1.r_i32",
  "exp": "val > 0"
}
```

应答 2

```
{
  "response": "sub_countruntime",
  "station": "sub1",
  "seqno": 2018,
  "status": 0,
  "runtimes": 12489370,
  "recordcounts": 10,
  "message": "OK"
}
```

3.8 值标签变化统计 rdb_countkst

用来统计开关变位次数。**rdb_countkst** 从主站统计，**sub_countkst** 从子站统计。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_countkst"从主站统计 "sub_countkst"从子站统计。
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_countkst"时有效
tag	string	标签名, 对于子站可以前面加子站名用冒号分开, 应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	String/number	开始日期时间, string 或者 number, 参见 2.3 时标
time_end	String/number	结束日期时间, string 或者 number, 参见 2.3 时标, 不提供此字段表示没有结束时间一直读完
lowval	int	低位值
hightval	int	高位值

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_countkst"从主站统计 "sub_countkst"从子站读取
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_countkst"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
l2hcounts	int	由低位变为高位的次数。
h2lcounts	int	由高位变为低位的次数。

例子:

请求 1 从主站统计

```
{  
    "request": "rdb_countkst",  
    "seqno": 2019,
```

```

        "time_begin": "2020/2/1",
        "time_end": "2020/3/1",
        "tag": "opc1.r_i32",
        "lowval": 0,
        "hightval": 1
    }

```

应答 1

```

{
    "response": "rdb_countkst",
    "seqno": 2019,
    "status": 0,
    "message": "OK",
    "l2hcounts": 2,
    "h2lcounts": 2
}

```

请求 2: 从子站统计

```

{
    "request": "sub_countkst",
    "seqno": 2020,
    "station": "sub1",
    "time_begin": "2020/2/1",
    "time_end": "2020/3/1",
    "tag": "opc1.r_i32",
    "lowval": 0,
    "hightval": 1
}

```

应答 2

```

{
    "response": "sub_countkst",
    "station": "sub1",
    "seqno": 2020,
    "status": 0,
    "message": "OK",
    "l2hcounts": 2,
    "h2lcounts": 2
}

```


3.9 读取越限标签快照 rdb_getoutlimitsnaps

读取当前处于越限状态的标签快照。

rdb_getoutlimitsnaps 从主站读取，**sub_getoutlimitsnaps** 通过主站代理读取指定子站。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_getoutlimitsnaps"读取主站标签越限快照 "sub_getoutlimitsnaps"读取子站标签越限快照
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_getoutlimitsnaps"时有效

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_getoutlimitsnaps"读取主站标签越限快照 "sub_getoutlimitsnaps"读取子站标签越限快照
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_getoutlimitsnaps"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	记录集，JSON 对象数组，每个对象字段定义： "ALARM": string 越限状态，下面 4 个字符串之一 "LIMIT_LOW", "LIMIT_LOLOW", "LIMIT_HIGH", "LIMIT_HIGHHIGH" "N" : string; 标签名; "DT": int ;数据类型, DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7; "Q" : int; 数据质量;无此字段或者 0 表示 good "T" : string 或者 number,参见 2.3 时标。

		"V" : 标签值
--	--	-----------

例子 1 直接读取:

请求:

```
{
  "request": "rdb_getoutlimitsnaps",
  "seqno": 3080
}
```

应答:

```
{
  "response": "rdb_getoutlimitsnaps",
  "seqno": 3080,
  "status": 0,
  "message": "OK",
  "vals": [{
    "ALARM": "LIMIT_HIGH",
    "N": "fr01",
    "DT": 3,
    "T": "2019-10-31 16:54:12.200",
    "Q": 0,
    "V": 200
  }, {
    "ALARM": "LIMIT_LOW",
    "N": "d0.dbl01.pv",
    "DT": 5,
    "T": "2020-06-29 16:37:23.000",
    "Q": 0,
    "V": 42.4924
  }
]
}
```

例子 2, 通过主站代理读取子站

请求:

```

{
    "request": "sub_getoutlimitsnaps",
    "station": "sub1",
    "seqno": 3081
}
应答:
{
    "response": "sub_getoutlimitsnaps",
    "station": "sub1",
    "seqno": 3081,
    "status": 0,
    "message": "OK",
    "vals": [{
        "ALARM": "LIMIT_HIGH",
        "N": "fr01",
        "DT": 3,
        "T": "2019-10-31 16:54:12.200",
        "Q": 0,
        "V": 200
    }, {
        "ALARM": "LIMIT_HIGH",
        "N": "d0.dbl01.pv",
        "DT": 5,
        "T": "2020-06-29 17:20:35.000",
        "Q": 0,
        "V": 55.0509
    }
    ]
}

```

3.10 订阅标签快照 rdb_sscsnaps

从连接的实时库订阅标签快照，当有新快照数据时会主动通过 **rdb_pushsnaps** 命令推送给订阅者，自 **RDB2024.9 inver5119** 版开始支持。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_sscsnaps"
seqno	int	>0，客户端填写，服务端原样返回

tags	object[]	<p>订阅标签对象数组，每个对象包含标签名和订阅模式两个字段。</p> <p>例：</p> <pre>[{ "name": "tst.f11", "mode": 1 }, { "name": "tst.f21", "mode": 1 }]</pre> <p>name 字段可以是通配字符串和精确标签名，通配标签(指带有*和?匹配的字符串，比如"d11.*")，如果要订阅全部标签，可以使用"*"。</p> <p>其中 mode 定义：</p> <p>0：订阅后不推送当前快照值</p> <p>1：订阅后推送当前快照值。</p> <p>当前快照是指订阅前已经存在的快照。mode 值不影响订阅后新产生的快照的推送行为。</p>
------	----------	--

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_sscsnaps "
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。
tags	object[]	<p>错误的订阅标签对象数组，如果没有错误，是一个空数组。</p> <p>例子：</p> <pre>[{ "name": "tst.f11", "mode": 1, "status": 48 }, { "name": "tst.f21", "mode": 1, "status": 48 }]</pre>

	其中 status 表示错误码， 48 为标签不存在。通配标签不会有失败。
--	---

例：订阅请求

```
{
  "request": "rdb:sscsnaps",
  "seqno": 3133,
  "tags": [
    {
      "name": "d11.*",
      "mode": 1
    },
    {
      "name": "tst.f21",
      "mode": 1
    },
    {
      "name": "tst.str1",
      "mode": 1
    }
  ]
}
```

订阅应答：

```
{
  "response": "rdb:sscsnaps",
  "seqno": 3133,
  "status": 0,
  "tags": [
    {
      "name": "tst.f21",
      "mode": 1,
      "status": 48
    },
    {
      "name": "tst.str1",
      "mode": 1,
      "status": 48
    }
  ]
}
```

}

3.11 取消标签快照订阅 rdb_unsscsnaps

是订阅的逆操作，取消后，服务端不再推送该标签的快照，自 RDB2024.9 inver5119 版开始支持。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_unsscsnaps"
seqno	int	>0，客户端填写，服务端原样返回
tags	object[]	需要取消订阅的标签对象数组，每个对象包含标签名一个字段。 例： [{ "name": "tst1.f11" }, { "name": "tst2.f21" }] name 字段可以是通配字符串和精确标签名，通配标签(指带有*和?匹配的字符串，比如"d11.*")。

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_unsscsnaps "
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。
tags	object[]	错误的订阅标签对象数组，如果没有错误，是一个空数组。 例子： [{ "name": "tst1.f11", "status": 76 }, { "name": "tst2.f21", "status": 76 }]

		<pre> }]</pre> <p>其中 status 表示错误码，76 表示该标签之前没有被订阅，48 表示标签不存在。</p>
--	--	--

例：取消订阅请求

```
{
  "request": "rdb_unsscsnaps",
  "seqno": 3035,
  "tags": [
    {
      "name": "d0.*"
    },
    {
      "name": "tst22.f2"
    }
  ]
}
```

取消订阅应答：

```
{
  "response": "rdb_unsscsnaps",
  "seqno": 3035,
  "status": 0,
  "tags": [
    {
      "name": "tst22.f2",
      "status": 48
    }
  ]
}
```

3.12 取消所有标签快照订阅 rdb_unsscallsnaps

取消当前连接的全部标签快照订阅，服务端不再推送标签快照，自 RDB2024.9 **inver5119** 版开始支持。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_unsscallsnaps"
seqno	int	>0, 客户端填写, 服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_unsscallsnaps"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。

例: 取消所有订阅请求

```
{"request": "rdb_unsscallsnaps", "seqno": 3036}
```

取消所有应答请求:

```
{
    "response": "rdb_unsscallsnaps",
    "seqno": 3036,
    "status": 0,
    "message": "OK"
}
```

3.13 读取标签快照订阅表 rdb_listsscsnaps

读取当前连接的全部标签快照订阅表, 自 RDB2024.9 inver5119 版开始支持。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_listsscsnaps"
seqno	int	>0, 客户端填写, 服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_listsscsnaps"
seqno	int	>0, 客户端填写, 服务端原样返回

status	int	0 表示成功，其余为错误码。
ucid	int	当前连接的会话 ID
tags	Object[]	订阅对象数组同 rdb_pushsnaps 请求消息的 tags。

例：读取订阅列表

```
{"request": "rdb_listsscsnaps", "seqno": 3039}
```

应答读取订阅列表

```
{
  "response": "rdb_listsscsnaps",
  "seqno": 3039,
  "status": 0,
  "ucid": 1002,
  "tags": [
    {
      "name": "d0.*",
      "mode": 1
    },
    {
      "name": "tst.f2",
      "mode": 1
    }
  ]
}
```

3.14 快照推送 rdb_pushsnaps

当有匹配订阅表的快照数据时，实时库会主动推送快照数据报文，自 RDB2024.9 inver5119 版开始支持。

推送报文字段说明

KEY	VAL 类型	说明
response	string	"rdb_pushsnaps"
seqno	int	0 始终为 0
status	int	0 始终为 0
vals	Object[]	标签快照对象数组，每个对象含有 5 个字段 "N" : string; 标签名; "DT": int ;数据类型,

		DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7; "Q" : int; 数据质量; 无此字段或者 0 表示 good "T" : string 或者 number, 参见 2.3 时标。 "V" : 标签值
--	--	---

例：推送快照报文

```
{
  "response": "rdb_pushsnaps",
  "seqno": 0,
  "status": 0,
  "vals": [
    {
      "N": "d0.k01.pv",
      "E": 0,
      "T": "2024-09-05T15:47:15.800+08:00",
      "Q": 1,
      "DT": 1,
      "V": 1
    },
    {
      "N": "d0.f01.pv",
      "E": 0,
      "T": "2024-09-19T10:32:08.100+08:00",
      "Q": 1,
      "DT": 3,
      "V": 2.5
    },
    {
      "N": "d0.str02.pv",
      "E": 0,
      "T": "2024-09-14T17:10:13.200+08:00",
      "Q": 0,
      "DT": 6,
      "V": "snapex test2"
    }
  ]
}
```

```

        "N": "d0.str01.pv",
        "E": 0,
        "T": "2024-09-15T00:35:58.000+08:00",
        "Q": 0,
        "DT": 6,
        "V": "snapex test"
    },
    {
        "N": "tst.f2",
        "E": 0,
        "T": "2024-09-04T09:45:30.100+08:00",
        "Q": 1,
        "DT": 3,
        "V": 2.5
    }
]
}

```

3.15 统计增量 rdb_countincrement

用于读取累积量在一段时间内的增量，即结束时间处的值减去开始时间处的值。此命令 2025.4(inver 5123)版新增。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_countincrement"从主站统计 "sub_countincrement"从子站统计。
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_countincrement"时有效
tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	string	开始日期时间，string 或 number，参见 2.3 时标描述
time_end	string	结束日期时间，string 或 number，参见 2.3 时标描述，不提供此字段表示没有结束时间一直读完

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_countincrement"从主站统计

		"sub_countincrement"从子站读取
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_countincrement"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
rec_inc	object	增量记录, 时标为 time_begin, 起数据类型为标签的数据类型。

例子: 读取一小时增量, 假设 d0.f01.pv 是累计值标签。

请求:

```
{
  "request": "rdb_countincrement",
  "seqno": 2022,
  "tag": "d0.f01.pv",
  "time_begin": "2025-03-01T01:0:0.000+08:00",
  "time_end": "2025-03-01T02:0:0.000+08:00"
}
```

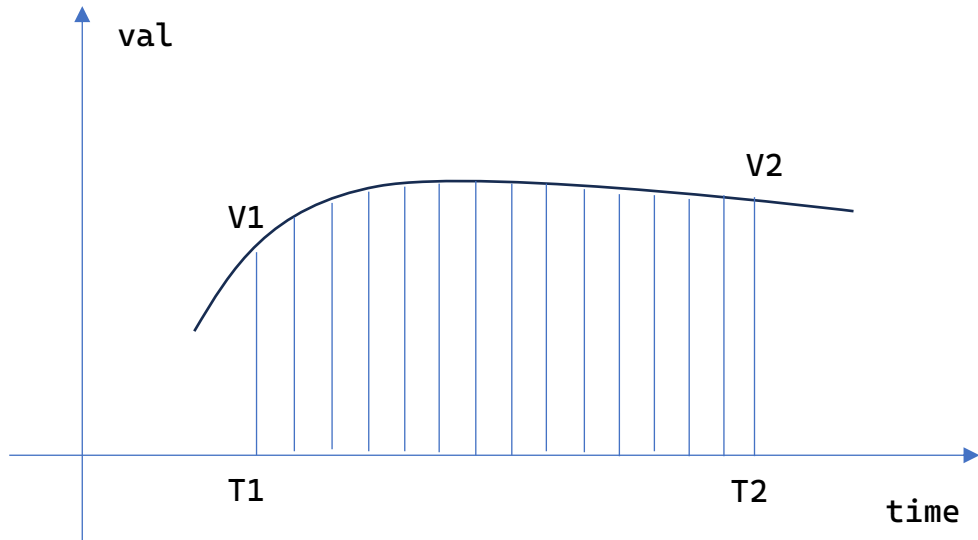
应答:

```
{
  "response": "rdb_countincrement",
  "seqno": 2022,
  "status": 0,
  "message": "OK",
  "rec_inc": {
    "T": "2025-03-01T01:00:00.000+08:00",
    "Q": 0,
    "DT": 3,
    "V": -0.116798
  }
}
```

3.16 统计积分累积量 rdb_countintegral

计算瞬时量在时间段内的积分累积量，比如将从 **t1** 到 **t2** 时间段内，不断动态变化的吹氧
气流量升/分钟统计计算为累积量，一共吹了多少升氧气。此命令 **2025.4(inver 5123)**
版新增。

积分原理：



积分累计值就是上图(T1,V1),(T2,V2)曲线和时间轴组成的阴影的面积。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_countintegral"从主站统计 "sub_countintegral"从子站统计。
seqno	int	>0, 客户端填写，服务端原样返回
station	string	子站名，request="sub_countintegral"时有效
tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时 和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	string	开始日期时间，string 或 number，参见 2.3 时标描述
time_end	string	结束日期时间，string 或 number，参见 2.3 时标描 述，不提供此字段表示没有结束时间一直读完
timeunit	int	瞬时量时间单位秒数； 升/分钟就填 60；立方/小时就填 3600

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_countintegral"从主站统计 "sub_countintegral"从子站读取
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_countintegral"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
integralval	double	积分累积量。

例子: 读取一小时积分累计, 假设 d0.f01.pv 是瞬时值标签, 单位为 升/分钟。

请求:

```
{
  "request": "rdb_countintegral",
  "seqno": 2022,
  "tag": "d0.f01.pv",
  "time_begin": "2025-03-01T01:0:0.000+08:00",
  "time_end": "2025-03-01T02:0:0.000+08:00",
  "timeunit": 60
}
```

应答:

```
{
  "response": "rdb_countintegral",
  "seqno": 2022,
  "status": 0,
  "message": "OK",
  "integralval": 118.860344
}
```

4 数据写入删除类命令详解

本章描述的写入快照，补录历史，历史删除，写入现场设备(控制输出)命令。

4.1 写入快照 rdb_writesnaps

用于数据网关提交实时数据，直接写入连接的实时库，不支持通过主站代理向子站写入。如果要写入子站，直接连接到子站写入。

提交值标签和对象标签均使用这个命令。注意写快照写入的时标必须大于实时库中已存在快照的时标，也就是时标必须是递增的，但是为了防止填写错误的时标，时标不能大于当前时间 24 小时。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_writesnaps"
seqno	int	>0, 客户端填写，服务端原样返回
vals	object array	记录集，JSON 对象数组，每个对象字段定义： "N" : string; 标签名; "E" : int ; 错误码; 0 表示成功，其他为错误码，写入时该字段可选。一般不填写。 "DT": int ;数据类型, DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7; "DT"也可用字符型来指示数据类型: "digital", "int32", "float", "double", "int64", "string", "object" "Q" : int; 数据质量; 无此字段或者 0 表示 good "T" : string 或者 number, 参见 2.3 时标 "V" : 标签值; 其中 object 对象标签使用 base64 编码

应答字段说明

KEY	VAL 类型	说明
-----	--------	----

response	string	"rdb_writesnaps"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	写入的错误记录集返回(正确的不返回), 如果某条记录写失败, 会在该条记录的"E"字段填写错误码。格式和写入时一样。

例子:

请求 1:

```
{
  "request": "rdb_writesnaps",
  "seqno": 2023,
  "vals": [
    {
      "N": "opc1.r_f32",
      "DT": 3,
      "T": "2020-03-02 17:50:57.000",
      "Q": 1,
      "V": 0.6257
    },
    {
      "N": "opc1.r_str",
      "DT": "string",
      "T": "2020-03-02 17:50:54.000",
      "Q": 0,
      "V": "2020-03-02 17:50:54"
    },
    {
      "N": "opc1.r_blob",
      "DT": "object",
      "T": "2020-03-02 17:50:54.000",
      "Q": 0,
      "V": "MjAyMC0wMy0wMiAxNzo1MDo1NA=="
    }
  ]
}
```

应答 1:


```

{
  "response": "rdb_writesnaps",
  "seqno": 2023,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "N": "opc1.r_f32",
      "E": 48,
      "DT": 3,
      "T": "2020-03-02 17:50:57.000",
      "Q": 1,
      "V": 0.6257
    },
    {
      "N": "opc1.r_str",
      "E": 48,
      "DT": 6,
      "T": "2020-03-02 17:50:54.000",
      "Q": 0,
      "V": "2020-03-02 17:50:54"
    },
    {
      "N": "opc1.r_blob",
      "E": 48,
      "DT": 7,
      "T": "2020-03-02 17:50:54.000",
      "Q": 0,
      "V": "MjAyMCOwMy0wMiAxNzo1MDo1NA=="
    }
  ]
}

```

4.2 补录历史 rdb_insert

用于数据网关补录历史数据，直接写入连接的实时库，不支持通过主站代理向子站写入。如果要写入子站，直接连接到子站写入。

补录值标签和对象标签均使用这个命令。这个命令和 **rdb_writesnaps** 的用法完全一样。

请求字段说明

KEY	VAL 类型	说明
-----	--------	----

request	string	"rdb_insert"
seqno	int	>0, 客户端填写, 服务端原样返回
vals	object array	<p>记录集, JSON 对象数组, 每个对象字段定义:</p> <p>"N" : string; 标签名;</p> <p>"E" : int ; 错误码; 0 表示成功, 其他为错误码, 写入时该字段可选。一般不填写。</p> <p>"DT": int ;数据类型,</p> <p>DIGITAL=1;</p> <p>INT32=2;</p> <p>FLOAT=3;</p> <p>INT64=4;</p> <p>DOUBLE=5;</p> <p>STRING=6;</p> <p>OBJECT=7;</p> <p>"DT"也可用字符型来指示数据类型:</p> <p>"digital", "int32", "float", "double", "int64", "string", "object"</p> <p>"Q" : int; 数据质量; 无此字段或者 0 表示 good</p> <p>"T" : string 或者 number, 参见 2.3 时标。</p> <p>"V" : 标签值; 其中 object 对象标签使用 base64 编码</p>

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_insert"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	写入错误的记录集返回(正确的不返回), 如果某条记录写失败, 会在该条记录的"E"字段填写错误码。格式和写入时一样。

例子:

请求 1

```
{
  "request": "rdb_insert",
  "seqno": 2024,
  "vals": [
```

```

    {
        "N": "opc1.r_f32",
        "DT": 3,
        "T": "2020-03-02 16:50:58.000",
        "Q": 1,
        "V": 0.6257
    },
    {
        "N": "opc1.r_str",
        "DT": "string",
        "T": "2020-03-02 16:50:55.000",
        "Q": 0,
        "V": "2020-03-02 16:50:55"
    },
    {
        "N": "opc1.r_blob",
        "DT": "object",
        "T": "2020-03-02 16:50:55.000",
        "Q": 0,
        "V": "MjAyMC0wMy0wMiAxNjo1MDo1NQ=="
    }
]
}

```

应答 1

```

{
    "response": "rdb_insert",
    "seqno": 2024,
    "status": 0,
    "message": "OK",
    "vals": [
        {
            "N": "opc1.r_f32",
            "E": 48,
            "DT": 3,
            "T": "2020-03-02 16:50:58.000",
            "Q": 1,
            "V": 0.6257
        },
        {
            "N": "opc1.r_str",
            "E": 48,

```

```

        "DT": 6,
        "T": "2020-03-02 16:50:55.000",
        "Q": 0,
        "V": "2020-03-02 16:50:55"
    },
    {
        "N": "opc1.r_blob",
        "E": 48,
        "DT": 7,
        "T": "2020-03-02 16:50:55.000",
        "Q": 0,
        "V": "MjAyMC0wMy0wMiAxNjo1MDo1NQ=="
    }
]
}

```

4.3 写设备(控制输出)rdb_writedevic/sub_writedevic

rdb_writedevic 写入挂接到主站数据网关的设备，**sub_writedevic** 写入挂接到子站数据网关的设备。每次写一个标签的一个值，值标签和对象标签均使用这个命令。

sub_writedevic 需要带上 **station** 字段。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_writedevic"
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_writedevic"时有效
vals	object	<p>一个标签值记录, 用一个 JSON 对象描述, 只有 3 个字段, 每个对象字段定义:</p> <p>"N" : string; 标签名;</p> <p>"DT": int ;数据类型,</p> <p>DIGITAL=1;</p> <p>INT32=2;</p> <p>FLOAT=3;</p> <p>INT64=4;</p> <p>DOUBLE=5;</p> <p>STRING=6;</p> <p>OBJECT=7;</p> <p>"DT"也可用字符型来指示数据类型:</p> <p>"digital", "int32", "float", "double",</p> <p>"int64", "string", "object"</p>

		"V" : 标签值; 其中 object 对象标签使用 base64 编码
--	--	---

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_writedevicel"
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_writedevicel"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。

例子

请求 1: 写主站设备值标签

```
{
  "request": "rdb_writedevicel",
  "seqno": 2025,
  "vals": {
    "N": "opc1.w_i32",
    "DT": "int32",
    "V": 1
  }
}
```

应答 1

```
{
  "response": "rdb_writedevicel",
  "seqno": 2025,
  "status": 0,
  "message": "OK"
}
```

请求 2: 写主站设备对象标签

```
{
  "request": "rdb_writedevicel",
  "seqno": 2026,
  "vals": {
    "N": "opc1.w_blob",
    "DT": "object",

```

```

        "V": "MjAyMC0wMy0wMiAxNjo1MDo1NQ=="
    }
}

```

应答 2:

```

{
    "response": "rdb_writedevic",
    "seqno": 2026,
    "status": 0,
    "message": "OK"
}

```

请求 3 写子站设备

```

{
    "request": "sub_writedevic",
    "seqno": 2027,
    "station": "sub1",
    "vals": {
        "N": "opc1.w_str",
        "DT": "string",
        "V": "write device test string"
    }
}

```

应答 3

```

{
    "response": "sub_writedevic",
    "station": "sub1",
    "seqno": 2027,
    "status": 0,
    "message": "OK"
}

```

4.4 人工置数 rdb_manual_set/sub_manual_set

rdb_manual_set 设置中心站或者直接连接的子站内标签，**sub_manual_set** 通过中心站代理设置子站内标签。设置人工置数的标签不在接受快照数据写入，直到通过 **rdb_manual_del/sub_manual_del** 取消人工置数。人工置数的数据质量定义为 5。

注意使用 **sub_manual_set** 需要带上 **station** 字段。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_manual_set"
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_manual_set"时有效
vals	object	<p>一个标签值记录, 用一个 JSON 对象描述, 只有 3 个字段, 每个对象字段定义:</p> <p>"N" : string; 标签名;</p> <p>"DT": int ;数据类型,</p> <p>DIGITAL=1;</p> <p>INT32=2;</p> <p>FLOAT=3;</p> <p>INT64=4;</p> <p>DOUBLE=5;</p> <p>STRING=6;</p> <p>OBJECT=7;</p> <p>"DT"也可用字符型来指示数据类型:</p> <p>"digital", "int32", "float", "double", "int64", "string", "object"</p> <p>"V" : 标签值; 其中 object 对象标签使用 base64 编码</p>

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_manual_set"
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_manual_set"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。

例子 1, 直连实时库操作。

```
{
  "request": "rdb_manual_set",
  "seqno": 3025,
  "vals": {
    "N": "d0.k00.pv",
    "DT": "digital",
    "V": 1
  }
}
```

```
    }  
}
```

应答:

```
{  
  "response": "rdb_manual_set",  
  "seqno": 3025,  
  "status": 0,  
  "message": "OK"  
}
```

例子 2: 通过中心站代理操作

```
{  
  "request": "sub_manual_set",  
  "seqno": 3026,  
  "station": "sub1",  
  "vals": {  
    "N": "d0.k00.pv",  
    "DT": "digital",  
    "V": 1  
  }  
}
```

应答:

```
{  
  "response": "sub_manual_set",  
  "seqno": 3026,  
  "station": "sub1",  
  "status": 0,  
  "message": "OK"  
}
```

4.5 取消人工置数 rdb_manual_del/sub_manual_del

rdb_manual_del 操作直接连接的实时库内标签, **sub_manual_del** 通过中心站代理操作子站内标签。取消成功后接受快照数据的写入。

请求字段说明

KEY	VAL 类型	说明
-----	--------	----

request	string	"rdb_manual_del"
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_manual_del"时有效
tagname	string	标签名

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_manual_del"
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_manual_del"时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。

例子 1, 直连实时库操作。

请求:

```
{
  "request": "rdb_manual_del",
  "seqno": 3036,
  "tagname": "d0.k00.pv"
}
```

应答:

```
{
  "response": "rdb_manual_del",
  "seqno": 3036,
  "status": 0,
  "message": "OK"
}
```

例子 2, 通过中心站代理

请求:

```
{
  "request": "sub_manual_del",
  "seqno": 3037,
  "station": "sub1",
```

```

    "tagname": "d0.k00.pv"
}

```

应答:

```

{
    "response": "sub_manual_del",
    "seqno": 3037,
    "station": "sub1",
    "status": 0,
    "message": "OK"
}

```

4.6 注册控制标签 rdb_regctrltag

用于支持控制输出的客户端，比如 **rdbdac_ux** 数据网关，向实时库注册该标签的下传控制，当其他客户端有该标签的控制输出是，实时库会把控制命令转发到这个注册该标签的客户端(或 **rdbdac_ux**)。每次调用注册一个标签。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_regctrltag"
seqno	int	>0 ，客户端填写，服务端原样返回
tag	string	标签名

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_regctrltag"
seqno	int	>0 ，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。

例子 1。

请求:

```

{
    "request": "rdb_regctrltag",
    "seqno": 3037,
    "tag": "d0.k00.pv"
}

```

应答:

```
{
    "response": "rdb_regctrltag",
    "seqno": 3037,
    "status": 0,
    "message": "OK"
}
```

4.7 控制输出 ctrl_device

实时库转发的控制消息，客户端(一般是 rdbdac_ux)收到这个消息后，将数据输出到具体设备。该命令无需应答。

请求字段说明

KEY	VAL 类型	说明
request	string	"ctrl_device"
seqno	int	0, 始终为 0
vals	object	一个标签的数据记录,只需要"N", "DT", "V"三个字段。

例子：实时库推送的命令

```
{
    "response": "rdb_device",
    "seqno": 3038,
    "status": 0,
    "message": "OK",
    "vals":
        {
            "N": "opc1.w_f32",
            "DT": 3,
            "V": 0.6257
        }
}
```

4.8 无压缩插入历史 rdb_insertex

无压缩方式插入历史，包括基本类型和字符串对象类型标签历史数据记录，时标相同时内容不同会更新记录，直接写入连接的实时库，不能通过主站代理向子站写入。如果要写入子站，直接连接到子站写入。(2023.4 版，内部版本 5105 开始支持本命令)

插入的时同一标签的多条记录，每条记录没有 N 字段。这个命令主要有以下三种用途：

- 1) 用于存储一些不变的配置值，固定时标，每次写入为更新模式。

- 2) 用于自己实现数据的备份模式写入。
- 3) 用于数据的精准维护，比如要精准插入一条样本值。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_insertex"
seqno	int	>0, 客户端填写, 服务端原样返回
tagname	String	标签名, 注意后面 vals 数组都是该标签的数据。
vals	object array	<p>记录集, JSON 对象数组, 每个对象字段定义:</p> <p>"DT": int ;数据类型, DIGITAL=1; INT32=2; FLOAT=3; INT64=4; DOUBLE=5; STRING=6; OBJECT=7;</p> <p>"DT"也可用字符型来指示数据类型: "digital", "int32", "float", "double", "int64", "string", "object"</p> <p>"Q" : int; 数据质量; 无此字段或者 0 表示 good "T" : string 或者 number, 参见 2.3 时标。 "V" : 标签值; 其中 object 对象标签使用 base64 编码</p> <p>每次提交记录数不大于 1000 条, 否则会返回错误。</p>

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_insertex"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
num	int	处理的记录数
num_err	int	错误记录数
num_insert	int	插入记录数

num_update	int	更新记录数
------------	-----	-------

例子:

请求 1

```
{
  "request": "rdb_insertex",
  "seqno": 2024,
  "tagname": "opc1.r_f32",
  "vals": [ {
    "DT": 3,
    "T": "2020-03-02 16:50:58.000",
    "Q": 0,
    "V": 0.6257
  },
  {
    "DT": 3,
    "T": "2020-03-02 16:53:58.000",
    "Q": 0,
    "V": 0.635
  }
]
}
```

应答 1

```
{
  "response": "rdb_insertex",
  "seqno": 2024,
  "status": 0,
  "message": "OK",
  "num": 2,
  "num_err": 0,
  "num_insert": 2,
  "num_update": 0
}
```

4.9 删除标签数据 rdb_datadelete

用于删除指定标签指定时间段的数据。(2023.4 版, 内部版本 5105 开始支持本命令)

请求字段说明

KEY	VAL 类型	说明
-----	--------	----

request	string	"rdb_datadelete"
seqno	int	>0, 客户端填写, 服务端原样返回
tagname	string	标签名
time_begin	string/number	开始时间(含), string 或者 number, 参见 2.3 时标
time_end	string/number	结束时间(不含), string 或者 number, 参见 2.3 时标; 不提供此字段或者填写-1 表示从开始时标之后的数据全部删除。

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_datadelete"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
records	Number	删除的记录数

例子 1, 删除一个月数据 。

请求:

```
{
  "request": "rdb_datadelete",
  "seqno": 3037,
  "tag": "d4.f01.pv",
  "time_begin": "2023-1-1T0:0:0.000+08:00",
  "time_end" : "2023-2-1T0:0:0.000+08:00"
}
```

应答:

```
{
  "response": "rdb_datadelete",
  "seqno": 3037,
  "status": 0,
  "message": "OK",
  "records": 23204
}
```

4.10 读取当前工作模式 rdb_getworkmode

工作模式默认为 **work** 模式，从 **2023.4** 内部版本 **5105** 开始增加 **manage** 模式；

实时库单独部署时，**work** 模式和 **manage** 模式完全一样，没有区别；

双机热备部署时，在默认的 **work** 模式下，以下命令对备机的请求会转发到主机处理：

```
rdb_getsnap  
rdb_getoutlimitsnaps  
rdb_valgetsection  
rdb_objget  
rdb_valquery  
rdb_plotdata  
rdb_sum  
rdb_countkst  
rdb_countruntime  
rdb_soequery
```

设置为 **manage** 模式就可以强制不转发，只对连接的服务器操作。可以避免误会，比如已经使用 **rdb_datadelete** 命令删除了备机上的数据，结果在 **work** 模式还能查到数据。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_getworkmode"
seqno	int	>0，客户端填写，服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_ getworkmode "
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
timefmt	string	返回当前客户端登录时协商的时标格式。

workmode	string	工作模式
-----------------	---------------	------

例子 1 读取当前工作模式

请求:

```
{"request": "rdb_getworkmode", "seqno": 3039}
```

应答:

```
{
  "response": "rdb_getworkmode",
  "seqno": 3039,
  "status": 0,
  "message": "OK",
  "timefmt": "iso",
  "workmode": "work"
}
```

4.11 设置工作模式 rdb_setworkmode

工作模式默认为 **work** 模式，从 2023.4 内部版本 5105 开始增加 **manage** 模式；

实时库单独部署时，**work** 模式和 **manage** 模式完全一样，没有区别；

双机热备部署时，在默认的 **work** 模式下，以下命令对备机的请求会转发到主机处理：

```
rdb_getsnap
rdb_getoutlimitsnaps
rdb_valgetsection
rdb_objget
rdb_valquery
rdb_plotdata
rdb_sum
rdb_countkst
rdb_countruntime
rdb_soequery
```

设置为 **manage** 模式就可以强制不转发，只对连接的服务器操作。可以避免误会，比如已经使用 **rdb_datadelete** 命令删除了备机上的数据，结果在 **work** 模式还能查到数据。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_setworkmode"
seqno	int	>0, 客户端填写, 服务端原样返回
workmode	string	"work"或者"manage"之一

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_setworkmode "
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
timefmt	string	返回当前客户端登录时协商的时标格式。
workmode	string	工作模式

例子 1, 设置为 **manage** 模式

请求:

```
{"request": "rdb_setworkmode", "seqno": 3040, "workmode": "manage"}
```

应答:

```
{
  "response": "rdb_setworkmode",
  "seqno": 3040,
  "status": 0,
  "message": "OK",
  "timefmt": "iso",
  "workmode": "manage"
}
```

注: 也可以在登录命令里加上 **workmode** 字段指定登录后的工作模式。

5 标签管理命令详解

本章简述标签的添加，修改，删除。读取和查询请见 3.1 和 3.2。对标签的操作需要连接到目标实时库，不能由主站代理。

5.1 读取标签属性 rdb_tagget/sub_tagget

rdb_tagget 从读取主站标签属性，**sub_tagget** 从主站代理读取子站标签属性。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_tagget"读取主站标签属性 "sub_tagget"读取子站标签属性
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名， request ="sub_tagget"时有效
tags	string array	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_f32" 和 "opc1.r_f32" 等价

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_tagget"读取主站标签属性 "sub_tagget"读取子站标签属性
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名， request ="sub_tagget"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	记录集，JSON 对象数组，每个对象字段定义具体参见下面例子，如果没有读到该标签则 errcode 非 0 时，其他字段为全 0 且没有意义。 sres 是保留的用 base64 编码的二进制信息。

例子：

请求 1(读取主站标签属性)

```
{  
    "request": "rdb_tagget",
```

```

    "seqno": 2005,
    "tags": [
        "opc1.r_f32",
        "opc1.r_str",
        "opc1.r_blob",
        "tagnotexist"
    ]
}

```

应答 1

```

{
    "response": "rdb_tagget",
    "seqno": 2005,
    "status": 0,
    "message": "OK",
    "vals": [
        {
            "name": "opc1.r_f32",
            "errcode": 0,
            "des": "1#机组温度",
            "unit": "°C",
            "datatype": 3,
            "tagtype": 0,
            "comptype": 1,
            "compval": 0.1,
            "compmin": 0,
            "compmax": 3600,
            "excdev": 0,
            "excmin": 0,
            "excmax": 0,
            "digits": 3,
            "class": 0,
            "step": 0,
            "archive": 1,
            "dnval": 0,
            "upval": 0,
            "alarmtype": 0,
            "alarm_llv": 0,
            "alarm_lv": 0,
            "alarm_hv": 0,
            "alarm_hhv": 0,

```

```

        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    },
    {
        "name": "opc1.r_str",
        "errcode": 0,
        "des": "",
        "unit": "",
        "datatype": 6,
        "tagtype": 0,
        "comptype": 0,
        "compval": 0,
        "compmin": 0,
        "compmax": 3600,
        "excdev": 0,
        "excmin": 0,
        "excmax": 0,
        "digits": 0,
        "class": 0,
        "step": 0,
        "archive": 1,
        "dnval": 0,
        "upval": 0,
        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    },
    {
        "name": "opc1.r_blob",
        "errcode": 0,
        "des": "",
        "unit": "",
        "datatype": 7,
        "tagtype": 0,
        "comptype": 0,
        "compval": 0,
        "compmin": 0,
        "compmax": 3600,
        "excdev": 0,
        "excmin": 0,

```

```

    "excmax": 0,
    "digits": 0,
    "class": 0,
    "step": 0,
    "archive": 1,
    "dnval": 0,
    "upval": 0,
    "alarmtype": 0,
    "alarm_llv": 0,
    "alarm_lv": 0,
    "alarm_hv": 0,
    "alarm_hhv": 0,
    "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
},
{
    "name": "tagnotexist",
    "errcode": 48,
    "des": "",
    "unit": "",
    "datatype": 0,
    "tagtype": 0,
    "comptype": 0,
    "compval": 0,
    "compmin": 0,
    "compmax": 0,
    "excdev": 0,
    "excmin": 0,
    "excmax": 0,
    "digits": 0,
    "class": 0,
    "step": 0,
    "archive": 0,
    "dnval": 0,
    "upval": 0,
    "alarmtype": 0,
    "alarm_llv": 0,
    "alarm_lv": 0,
    "alarm_hv": 0,
    "alarm_hhv": 0,
    "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
}

```

```
]
}
```

请求 2 读取子站标签属性(标签名不带子站信息)

```
{
  "request": "sub_tagget",
  "seqno": 2006,
  "station": "sub1",
  "tags": [
    "opc1.r_f32",
    "opc1.r_str",
    "opc1.r_blob",
    "tagnotexist"
  ]
}
```

或者标签名带子站信息

```
{
  "request": "sub_tagget",
  "seqno": 2006,
  "station": "sub1",
  "tags": [
    "sub1:opc1.r_f32",
    "sub1:opc1.r_str",
    "sub1:opc1.r_blob",
    "sub1:tagnotexist"
  ]
}
```

应答 2

```
{
  "response": "sub_tagget",
  "station": "sub1",
  "seqno": 2006,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "opc1.r_f32",
      "errcode": 0,
      "des": "1#机组温度",
      "unit": "°C",
```

```

    "datatype": 3,
    "tagtype": 0,
    "comptype": 1,
    "compval": 0.1,
    "compmin": 0,
    "compmax": 3600,
    "excdev": 0,
    "excmin": 0,
    "excmax": 0,
    "digits": 3,
    "class": 0,
    "step": 0,
    "archive": 1,
    "dnval": 0,
    "upval": 0,
    "alarmtype": 0,
    "alarm_llv": 0,
    "alarm_lv": 0,
    "alarm_hv": 0,
    "alarm_hhv": 0,
    "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
},
{
    "name": "opc1.r_str",
    "errcode": 0,
    "des": "",
    "unit": "",
    "datatype": 6,
    "tagtype": 0,
    "comptype": 0,
    "compval": 0,
    "compmin": 0,
    "compmax": 3600,
    "excdev": 0,
    "excmin": 0,
    "excmax": 0,
    "digits": 0,
    "class": 0,
    "step": 0,
    "archive": 1,
    "dnval": 0,
    "upval": 0,

```

```

        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    },
    {
        "name": "opc1.r_blob",
        "errcode": 0,
        "des": "",
        "unit": "",
        "datatype": 7,
        "tagtype": 0,
        "comptype": 0,
        "compval": 0,
        "compmin": 0,
        "compmax": 3600,
        "excdev": 0,
        "excmin": 0,
        "excmax": 0,
        "digits": 0,
        "class": 0,
        "step": 0,
        "archive": 1,
        "dnval": 0,
        "upval": 0,
        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    },
    {
        "name": "tagnotexist",
        "errcode": 48,
        "des": "",
        "unit": "",
        "datatype": 0,
        "tagtype": 0,
        "comptype": 0,

```



```

        "compval": 0,
        "compmin": 0,
        "compmax": 0,
        "excdev": 0,
        "excmin": 0,
        "excmax": 0,
        "digits": 0,
        "class": 0,
        "step": 0,
        "archive": 0,
        "dnval": 0,
        "upval": 0,
        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    }
}
]
}

```

5.2 查询标签 rdb_tagquery/sub_tagquery

查询是通过提供标签名匹配串等类型等信息，查询匹配的标签列表。**rdb_tagquery** 从主站查询，**sub_tagquery** 从子站查询。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_tagquery" 查询主站标签属性 "sub_tagquery" 查询子站标签属性
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_tagquery" 时有效
name	string	可选，标签名匹配串，无此字段表示全部。支持*和?匹配
des	string	可选，标签描述匹配串，无此字段表示全部。支持*和?匹配
datatype	string	可选，数据类型字符串，无此字段表示全部，分别为 digital, int32, float, int64, double, string, object
class	int	可选，标签分类无此字段表示全部，分别为

		0 现场设备标签; 1 定义标签(手工标签); 2 预处理标签; 3 曲线标签; 4 系统标签
--	--	--

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_tagquery" 查询主站标签属性 "sub_tagquery" 查询子站标签属性
seqno	int	>0, 客户端填写, 服务端原样返回
station	string	子站名, request="sub_tagquery" 时有效
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	记录集, JSON 对象数组, 每个对象字段定义具体参见下面例子, sres 是保留的用 base64 编码的二进制信息。

请求 1 查询主站标签

```
{
  "request": "rdb_tagquery",
  "seqno": 2008,
  "name": "opc1",
  "datatype": "int32"
}
```

应答 1

```
{
  "response": "rdb_tagquery",
  "seqno": 2008,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "opc1.r_i32",
      "errcode": 0,
      "des": "",
      "unit": "",
      "datatype": 2,

```

```

    "tagtype": 0,
    "comptype": 1,
    "compval": 0.1,
    "compmin": 0,
    "compmax": 3600,
    "excdev": 0,
    "excmin": 0,
    "excmax": 0,
    "digits": 0,
    "class": 0,
    "step": 0,
    "archive": 1,
    "dnval": 0,
    "upval": 0,
    "alarmtype": 0,
    "alarm_llv": 0,
    "alarm_lv": 0,
    "alarm_hv": 0,
    "alarm_hhv": 0,
    "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
},
{
    "name": "opc1.w_i32",
    "errcode": 0,
    "des": "",
    "unit": "",
    "datatype": 2,
    "tagtype": 0,
    "comptype": 1,
    "compval": 0.1,
    "compmin": 0,
    "compmax": 3600,
    "excdev": 0,
    "excmin": 0,
    "excmax": 0,
    "digits": 0,
    "class": 0,
    "step": 0,
    "archive": 1,
    "dnval": 0,
    "upval": 0,
    "alarmtype": 0,

```

```

        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    }
]
}

```

请求 2 查询子站标签

```

{
    "request": "sub_tagquery",
    "seqno": 2008,
    "station": "sub1",
    "name": "opc1",
    "datatype": "int32"
}

```

应答 2

```

{
    "response": "sub_tagquery",
    "station": "sub1",
    "seqno": 2008,
    "status": 0,
    "message": "OK",
    "vals": [
        {
            "name": "opc1.r_i32",
            "errcode": 0,
            "des": "",
            "unit": "",
            "datatype": 2,
            "tagtype": 0,
            "comptype": 1,
            "compval": 0.1,
            "compmin": 0,
            "compmax": 3600,
            "excdev": 0,
            "excmin": 0,
            "excmax": 0,
            "digits": 0,
            "class": 0,

```

```

        "step": 0,
        "archive": 1,
        "dnval": 0,
        "upval": 0,
        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    },
    {
        "name": "opc1.w_i32",
        "errcode": 0,
        "des": "",
        "unit": "",
        "datatype": 2,
        "tagtype": 0,
        "comptype": 1,
        "compval": 0.1,
        "compmin": 0,
        "compmax": 3600,
        "excdev": 0,
        "excmin": 0,
        "excmax": 0,
        "digits": 0,
        "class": 0,
        "step": 0,
        "archive": 1,
        "dnval": 0,
        "upval": 0,
        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    }
]
}

```

5.3 标签的添加和修改 rdb_taginport

添加和修改均使用这个命令。标签名作为唯一 ID，标签属性参见《rdb_user_guide.pdf》第 2.3 节描述。

每次可以添加和修改多个标签。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_taginport"
seqno	int	>0, 客户端填写, 服务端原样返回
mask	int	<p>修改属性时使用的掩码, 当标签存在时指示要修改那些项。如果标签不存在, 则不会使用这个 mask 值, 全部字段都用做新加标签。</p> <p>按照位定义的, 使用时转换为 10 进制。</p> <p>0x0001 //描述 0x0002 //工程单位 0x0004 //标签类型, ctagtype 字段 0x0008 //压缩属性 0x0010 //标签分类, TGCLS_DEC, 等 0x0020 //归档 0x0040 //显示精度 0x0080 //梯形插值 cstep 0x0100 //上下限值 0x0400 //告警信息 0x0800 //数据类型, cdatatype, 从内部版本 5122 开始支持基本数据类型可以修改。</p> <p>例子: 全部都要改填写 0x5FB = 1531; 或者填写 0xFFF=4095 也行。 只改描述和工程单位: 0x01 +0x02 = 3 只改压缩属性: 0x08 = 8</p>
vals	object array	<p>标签值属性记录集, JSON 对象数组, 每个标签属性的字段定义:</p> <p>"name" : string; 标签名; "errcode": int 可选, 错误码, 返回时采用, 默认值 0 表示成功, 非 0 为错误码。 "des": string ; 标签描述 "unit": string ; 工程单位 "datatype": int; 数据类型</p>

```

DIGITAL=1;
INT32=2;
FLOAT=3;
INT64=4;
DOUBLE=5;
STRING=6;
OBJECT=7;

```

"datatype"也可用字符型来指示数据类型:
 "digital", "int32", "float", "double",
 "int64", "string", "object"

"tagtype": int ,可选,标签类型,默认 0,客户端定义
 "comptype": int ,压缩方式

- 0 不压缩
- 1 百分比精度压缩
- 2 绝对值精度压缩
- 3 定时存储
- 4 增强定时存储

"compval": float, 压缩精度
 "compmin": int, 最小压缩周期(单位秒)
 "compmax": int, 最大压缩周期(单位秒)
 "excdev": float, 列外偏差,
 如果是百分比, 则和百分比压缩意义相同
 "excmmin": int, 最小例外周期(单位秒)
 "excmax": int, 最大例外周期(单位秒)
 "digits": int ,显示用小数位数,-20 到 0;
 大于 0 小数位数; 小于 0, 有效位数

"class": int 标签分类;可选,默认 默认 0

- 0 现场设备标签
- 1 定义标签(手工标签)
- 2 预处理标签
- 3 曲线标签
- 4 系统标签

"step": int , 可选, 未使用,1 梯形,0 线性,默认 0
 "archive": int 数据存盘归档,1 存盘,0 不存
 "dnval": float 工程值下限,应用层使用,可选,默认 0.
 "upval": float 工程值上限,应用层使用,可选,默认 0.
 "alarmtype": int 告警方式,应用层使用,可选,默认 0.
 "alarm_llv":float;低低限,应用层使用,可选,默认 0.
 "alarm_lv":float;低限,应用层使用,可选,默认 0.
 "alarm_hv":float;高限,应用层使用,可选,默认 0.

		"alarm_hhv":float;高高限,应用层使用,可选,默认 0.
--	--	--------------------------------------

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_taginport"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	<p>操作后的标签数组, 其中每个记录的</p> <p>errcode: 该标签的操作结果, 0 表示成功。</p> <p>sres 为 base64 编码, 解码后, sres[22]值表示操作结果</p> <p>0x00 : 未更改 0x01 : 修改 0x02 : 添加</p> <p>其余字段为添加和修改后实时库里标签的完整属性。</p>

例子

请求 1 添加两个标签

```
{
  "request": "rdb_taginport",
  "seqno": 2028,
  "vals": [
    {
      "name": "tst.t2.i32",
      "des": "test add tag",
      "datatype": "int32",
      "unit": "k",
      "comptype": 4,
      "compmax": 60
    },
    {
      "name": "tst.t2.f32",
      "des": "test add tag",
      "datatype": "float",
      "comptype": 1,
      "compval": 0.1,
      "compmax": 60
    }
  ]
}
```



```

    }
  ]
}

```

应答 1

```

{
  "response": "rdb_taginport",
  "seqno": 2028,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "tst.t2.i32",
      "errcode": 0,
      "des": "test add tag",
      "unit": "k",
      "datatype": 2,
      "tagtype": 0,
      "comptype": 4,
      "compval": 0,
      "compmin": 0,
      "compmax": 60,
      "excdev": 0,
      "excmin": 0,
      "excmax": 0,
      "digits": 0,
      "class": 0,
      "step": 0,
      "archive": 0,
      "dnval": 0,
      "upval": 0,
      "alarmtype": 0,
      "alarm_llv": 0,
      "alarm_lv": 0,
      "alarm_hv": 0,
      "alarm_hhv": 0,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    },
    {
      "name": "tst.t2.f32",
      "errcode": 0,
      "des": "test add tag",

```

```

        "unit": "",
        "datatype": 3,
        "tagtype": 0,
        "comptype": 1,
        "compval": 0.1,
        "compmin": 0,
        "compmax": 60,
        "excdev": 0,
        "excmin": 0,
        "excmax": 0,
        "digits": 0,
        "class": 0,
        "step": 0,
        "archive": 0,
        "dnval": 0,
        "upval": 0,
        "alarmtype": 0,
        "alarm_llv": 0,
        "alarm_lv": 0,
        "alarm_hv": 0,
        "alarm_hhv": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIA"
    }
]
}

```

请求 2 修改标签属性

将刚才添加的 **tst.t2.i32** 的工程单位改成 **"m/s"**, 最大压缩周期(定时存储周期)改为 **5 分钟(300 秒)**。

这时只提供 **unit** 和 **compmax** 字段, **mask** 为 **2+8=10**。

```

{
    "request": "rdb_taginport",
    "seqno": 2029,
    "mask": 10,
    "vals": [
        {
            "name": "tst.t2.i32",
            "unit": "m/s",
            "compmax": 300
        }
    ]
}

```

```
    ]
}
```

应答 2

```
{
  "response": "rdb_taginport",
  "seqno": 2029,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "tst.t2.i32",
      "errcode": 0,
      "des": "test add tag",
      "unit": "m/s",
      "datatype": 2,
      "tagtype": 0,
      "comptype": 0,
      "compval": 0,
      "compmin": 0,
      "compmax": 300,
      "excdev": 0,
      "excmin": 0,
      "excmax": 0,
      "digits": 0,
      "class": 0,
      "step": 0,
      "archive": 0,
      "dnval": 0,
      "upval": 0,
      "alarmtype": 0,
      "alarm_llv": 0,
      "alarm_lv": 0,
      "alarm_hv": 0,
      "alarm_hhv": 0,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAEA"
    }
  ]
}
```

5.4 删除标签 rdb_tagdel

删除标签，同时会删除实时库中该标签的历史数据，因此每次只提交一个标签删除。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_tagdel"
seqno	int	>0, 客户端填写，服务端原样返回
tag	string	需要删除的标签名

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_tagdel"
seqno	int	>0, 客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。

例子

请求 1 删除"tst.t2.i32"标签

```
{
    "request": "rdb_tagdel",
    "seqno": 2030,
    "tag": "tst.t2.i32"
}
```

应答 1:

```
{
    "response": "rdb_tagdel",
    "seqno": 2030,
    "status": 0,
    "message": "OK"
}
```

如果标签不存在，比如我们再删除一次，则会返回如下错误：

```
{
    "response": "rdb_tagdel",
    "seqno": 2030,
```

```

    "status": 48,
    "message": "tag not exist"
}

```

5.5 导入标签的扩展属性 rdb_taginport_ext

导入标签的扩展属性。扩展属性是和标签名名关联的，因此必须要先有标签才能导入其扩展属性。扩展属性目前定义有 SOE 的越限描述，用于服务端生成 SOE 越限事件用。

SOE 越限屏蔽码为 32768，包含以下 4 个属性(字段名)：

"des_ll"：越低低限的描述，小于 80 字节

"des_l"：越低限的描述，小于 80 字节

"des_h"：越高限的描述，小于 80 字节

"des_hh"：越高高限的描述，小于 80 字节

SOE 事件级别屏蔽码为 65536，包含以下 4 个属性(字段名)：

"lev_ll"：越低低限的级别，int

"lev_l"：越低限的级别，int

"lev_h"：越高限的级别，int

"lev_hh"：越高高限的级别，int

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_taginport_ext"
seqno	int	>0，客户端填写，服务端原样返回
mask	int	掩码，SOE 越限描述为 32768
vals	object array	标签扩展属性数组

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_tagdel"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
vals	object array	

例子

请求:

```
{
  "request": "rdb_taginport_ext",
  "seqno": 2002,
  "mask": 98304,
  "vals": [{
    "tagname": "d0.i01.pv",
    "des_ll": "越低低限",
    "des_l": "越低限",
    "des_h": "越高限",
    "des_hh": "越高高限",
    "lev_ll": 1,
    "lev_l": 2,
    "lev_h": 3,
    "lev_hh": 4
  }, {
    "tagname": "tagnoext.pv",
    "des_ll": "越低低限",
    "des_l": "越低限",
    "des_h": "越高限",
    "des_hh": "越高高限",
    "lev_ll": 1,
    "lev_l": 2,
    "lev_h": 3,
    "lev_hh": 4
  }
]
}
```

应答:

```
{
  "response": "rdb_taginport_ext",
  "seqno": 2002,
  "status": 0,
  "message": "OK",
  "vals": [{
```

```

        "tagname": "d0.i01.pv",
        "des_ll": "越低低限",
        "des_l": "越低限",
        "des_h": "越高限",
        "des_hh": "越高高限",
        "lev_ll": 1,
        "lev_l": 2,
        "lev_h": 3,
        "lev_hh": 4
    }, {
        "tagname": "tagnoext.pv",
        "status": 48,
        "des_ll": "越低低限",
        "des_l": "越低限",
        "des_h": "越高限",
        "des_hh": "越高高限",
        "lev_ll": 1,
        "lev_l": 2,
        "lev_h": 3,
        "lev_hh": 4
    }
]
}

```

如果标签不存在，则返回的对象会对一个 **status** 字段表示错误码

5.6 读取标签的扩展属性 rdb_tagget_ext

删除标签，同时会删除实时库中该标签的历史数据，因此每次只提交一个标签删除。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_tagget_ext"
seqno	int	>0，客户端填写，服务端原样返回
vals	string array	标签名字字符串数组

应答字段说明

KEY	VAL 类型	说明
-----	--------	----

response	string	"rdb_tagdel"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
vals	object array	

例子

请求:

```
{
  "request": "rdb_tagget_ext",
  "seqno": 2046,
  "vals": ["d0.i01.pv", "d0.i02.pv", "sys.i01.pv"]
}
```

应答:

```
{
  "response": "rdb_tagget_ext",
  "seqno": 2046,
  "status": 0,
  "message": "OK",
  "vals": [{
    "tagname": "d0.i01.pv",
    "des_ll": "越低低限",
    "des_l": "越低低",
    "des_h": "越高限",
    "des_hh": "越高高限",
    "lev_ll": 1,
    "lev_l": 2,
    "lev_h": 3,
    "lev_hh": 4
  }, {
    "tagname": "d0.i02.pv"
  }, {
    "tagname": "sys.i01.pv",
    "status": 48
  }
]
```


}

如果返回的对象数组中的对象只有标签名，表示没有扩展属性，如果有 **status** 字段则为错误码。

5.7 删除标签的扩展属性

在删除标签的同时，会删除该标签的全部扩展属性。

如果只是删除部分扩展属性，使用 **rdb_tagimport_ext** 将对应的字段设置为默认值即可，字符串字段默认值为空串，数字字段默认值为 0

5.8 导出标签表 rdb_tagexport

通过提供标签名匹配串等类型等信息，查询匹配的标签列表，按照指定的字符编码生成 **csv** 格式文件后使用 **bsae64** 编码应答。

此命令 **2022.7SP1** 版新增。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_tagexport" 查询主站标签属性
seqno	int	>0，客户端填写，服务端原样返回
name	string	可选，标签名匹配串，无此字段表示全部。支持*和?匹配
des	string	可选，标签描述匹配串，无此字段表示全部。支持*和?匹配
datatype	string	可选，数据类型字符串，无此字段表示全部，分别为 digital, int32, float, int64, double, string, object
class	int	可选，标签分类无此字段表示全部，分别为 0 现场设备标签； 1 定义标签(手工标签)； 2 预处理标签； 3 曲线标签； 4 系统标签
ecnode	string	"GBK", "UTF8"之一，不填默认 "GBK"

应答字段说明

KEY	VAL 类型	说明
-----	--------	----

6 账号管理命令详解

账号管理包括角色管理(增删改)，用户管理(增删改)。所有的账号管理操作都是操作直连的实时库，不能通过主站代理。

6.1 添加/修改角色 rdb_addactor

每次可以添加多个，存在则修改，请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_addactor"
seqno	int	>0，客户端填写，服务端原样返回
vals	object array	角色对象记录集，每个角色字段定义 "name": string 角色名, 小于 16 字节 "des" : string 描述 "power": int 权限，按位定义的权限，转换为 10 进制填写。 0x01 读取数据 0x02 写数据 0x04 写标签 0x08 账号管理 0x10 控制权限 比如全部权限就是 0x1F=31 "sres" : string , base64 编码的保留二进制数据

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_addactor"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	角色对象记录集，每个角色字段定义 "name": string 角色名 "des" : string 描述 "power": int 权限 int 权限，按位定义的权限，转换为 10 进制填写。

		<p> 0x01 读取数据 0x02 写数据 0x04 写标签 0x08 账号管理 0x10 控制权限 比如全部权限就是 0x1F=31 </p> <p> "errcode": int 操作错误码; 0 表示成功。 "sres" : string , base64 编码的保留二进制数据 </p>
--	--	--

例子

请求 1

```
{
  "request": "rdb_addactor",
  "seqno": 2033,
  "vals": [
    {
      "name": "testopt",
      "des": "test operator",
      "power": 24
    },
    {
      "name": "testopt2",
      "des": "test operator2",
      "power": 7
    }
  ]
}
```

应答 1

```
{
  "response": "rdb_addactor",
  "seqno": 2033,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "testopt",
      "des": "test operator",
      "power": 24,
      "errcode": 0,
    }
  ]
}
```

```

        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    },
    {
        "name": "testopt2",
        "des": "test operator2",
        "power": 7,
        "errcode": 0,
        "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    }
]
}

```

6.2 读取角色 rdb_listactor

读取全部角色列表。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_listactor"
seqno	int	>0, 客户端填写，服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_listactor"
seqno	int	>0, 客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	角色对象记录集，每个角色字段定义 "name": string 角色名 "des" : string 描述 "power": int 权限 int 权限，按位定义的权限，转换为 10 进制填写。 <div style="margin-left: 40px;"> 0x01 读取数据 0x02 写数据 0x04 写标签 0x08 账号管理 0x10 控制权限 </div>

		比如全部权限就是 0x1F=31
		"sres" : string , base64 编码的保留二进制数据

例子

请求 1

```
{
  "request": "rdb_listactor",
  "seqno": 2034
}
```

应答 1:

```
{
  "response": "rdb_listactor",
  "seqno": 2034,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "testopt",
      "des": "test operator",
      "power": 24,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    },
    {
      "name": "administrators",
      "des": "default administrator actor",
      "power": 31,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    },
    {
      "name": "operators",
      "des": "power for operators",
      "power": 23,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    },
    {
      "name": "testopt2",
      "des": "test operator2",
      "power": 7,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    }
  ]
}
```

```

    }
  ]
}

```

6.3 删除角色 rdb_delactor

删除一个角色，如果该角色已经被使用则删除失败。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_delactor"
seqno	int	>0，客户端填写，服务端原样返回
name	string	角色名

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_delactor"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。

例子

请求 1

```

{
  "request": "rdb_delactor",
  "seqno": 2035,
  "name": "testopt2"
}

```

应答 1

```

{
  "response": "rdb_delactor",
  "seqno": 2035,
  "status": 0,
  "message": "OK"
}

```

6.4 添加/修改账号 rdb_addoperator

每次可以添加多个，存在则修改，请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_addoperator"
seqno	int	>0，客户端填写，服务端原样返回
vals	object array	角色对象记录集，每个角色字段定义 "name": string 角色名,小于 16 字节 "pswd": string 密码 MD5 bin16bytes 的 base64 编码。 "des" : string 描述 "actor": string 角色名 "maxsessions" : int 该用户最大连接数, 0 或者不提高表示不限制（内部版本号 5.1.0.7 开始生效）。 "active" : int 是否活动有效; 1:有效(默认值) 0:禁用 "fixedpswd": int 固定密码; 可选 1:禁止修改; 0:可修改(默认值) "sres" : string ,可选 base64 编码的保留二进制数据

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_addoperator"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	角色对象记录集，每个角色字段定义 "name": string 账号名 "pswd": string 密码 MD5 bin16bytes 的 base64 编码。 "des" : string 描述 "actor": string 角色名 "maxsessions" : int 该用户最大连接数, 0 或者不提高表示不限制（内部版本号 5.1.0.7 开始生效）。 "active" : int 是否活动有效; 1:有效(默认值)

		0 :禁用 "fixedpswd": int 固定密码; 可选 1 :禁止修改; 0 :可修改(默认值) "sres" : string ,可选 base64 编码的保留二进制数据 "errcode" :int 操作结果; 0 :成功; 非 0 错误码。
--	--	---

例子

请求 1

```
{
  "request": "rdb_addoperator",
  "seqno": 2036,
  "vals": [
    {
      "name": "tstopt1",
      "pswd": "K7IlzrXuJJMWfP3g1y2iQg==",
      "actor": "operators",
      "maxsessions":10,
      "des": "test operators",
    }
  ]
}
```

应答 1

```
{
  "response": "rdb_addoperator",
  "seqno": 2036,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "tstopt1",
      "pswd": "K7IlzrXuJJMWfP3g1y2iQg==",
      "actor": "operators",
      "des": "test operators",
      "active": 1,
      "fixedpswd": 0,
      "maxsessions":10,
      "errcode": 0,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
```

```

    }
  ]
}

```

6.5 读取账号列表 rdb_listoperator

每次可以添加多个，存在则修改，请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_listoperator"
seqno	int	>0, 客户端填写，服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_listoperator"
seqno	int	>0, 客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	角色对象记录集，每个角色字段定义 "name": string 账号名 "pswd": string 密码 MD5 bin16bytes 的 base64 编码。 "des" : string 描述 "actor": string 角色名 "maxsessions" : int 该用户最大连接数，0 或者不提高表示不限制（内部版本号 5.1.0.7 开始生效）。 "active" : int 是否活动有效； 1:有效(默认值) 0:禁用 "fixedpswd": int 固定密码；可选 1:禁止修改； 0:可修改(默认值) "sres" : string ,可选 base64 编码的保留二进制数据

例子

请求 1

```

{
  "request": "rdb_listoperator",

```

```
    "seqno": 2037
}
```

应答 1

```
{
  "response": "rdb_listoperator",
  "seqno": 2037,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "name": "tstopt1",
      "pswd": "K7IlzrXuJJMWfP3g1y2iQg==",
      "actor": "operators",
      "maxsessions":10,
      "des": "test operators",
      "active": 1,
      "fixedpswd": 0,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    },
    {
      "name": "admin",
      "pswd": "ISMvKXpXpadDiUoOSoAfw==",
      "actor": "administrators",
      "des": "default administrator user ",
      "active": 1,
      "fixedpswd": 0,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    },
    {
      "name": "opt1",
      "pswd": "K7IlzrXuJJMWfP3g1y2iQg==",
      "actor": "operators",
      "maxsessions":10,
      "des": "test operators",
      "active": 1,
      "fixedpswd": 0,
      "sres": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=="
    }
  ]
}
```

6.6 删除账号 rdb_deloperator

删除一个操作员，不能删除 **admin**。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_deloperator"
seqno	int	>0，客户端填写，服务端原样返回
name	string	账号名

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_deloperator"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。

例子

请求 1

```
{
    "request": "rdb_deloperator",
    "seqno": 2038,
    "name": "tstopt1"
}
```

应答 1

```
{
    "response": "rdb_deloperator",
    "seqno": 2038,
    "status": 0,
    "message": "OK"
}
```

6.7 修改密码 rdb_modfypswd

修改账号密码，下次连接生效。

请求格式:

KEY	VAL 类型	说明
request	string	"rdb_modfypswd"
seqno	int	>0, 客户端填写, 服务端原样返回
name	string	账号名
oldpswd	string	旧密码, 密码 16 字节 MD5 散列值 base64 编码
newpswd	string	新密码, 密码 16 字节 MD5 散列值 base64 编码

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_modfypswd"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。

例子

请求 1

```
{
  "request": "rdb_modfypswd",
  "seqno": 2039,
  "name": "tstopt2",
  "oldpswd": "K7IlzrXuJJMWfP3g1y2iQg==",
  "newpswd": "ISMvKXpXpadDiUo0SoAfww=="
}
```

应答 1

```
{
  "response": "rdb_modfypswd",
  "seqno": 2039,
  "status": 0,
  "message": "OK"
}
```


7SOE 事件命令详解

SOE 事件单列一章，本章讲述事件的写入，查询，更新，订阅，接受推送。

7.1 查询 SOE 事件 rdb_soequery

rdb_soequery 从主站查询，sub_soequery 从子站查询。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_soequery"从主站统计 "sub_soequery"从子站统计。
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_soequery"时有效
tag	string	标签名，对于子站可以前面加子站名用冒号分开，应答时和请求的标签名格式一致。 比如 "sub1:opc1.r_str" 和 "opc1.r_str" 等价
time_begin	String/number	可选，开始日期时间，string 或者 number,参见 2.3 时标。
ukey	int	可选，默认 0，即时标相同时区别事件的 ID 号
time_end	String/number	可选，结束日期时间，string 或者 number,参见 2.3 时标。
exp	string	可选，查询表达式
top	int	可选，默认 100，最多读取记录数 < 256

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_soequery"从主站统计 "sub_soequery"从子站读取
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名，request="sub_soequery"时有效
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
end	int	1:结束；0: 没有结束，后面还有记录，可以再次修改 time_begin 和 ukey 继续读取。
vals	object array	SOE 记录集，JSON 对象数组，每个标签属性的字段定义：

	<p>"time" : string 或者 number, 参见 2.3 时标</p> <p>"ukey" : string : 该字段用于当时标重复时作为辅助 KEY 使用。</p> <p>"type": int 可选, 默认 0, 应用层定义的事件类型。</p> <p>"argtype": int, 可选, 默认 0: <0 为基于字符串的格式, >0 为基于 2 进制的格式。 具体有用户定义, 用于客户端解析参数。</p> <p>"arglen": int , 参数长度。可选, 默认 0 表示没有参数。</p> <p>"level": int, 事件级别, 可选, 默认 0, 应用层定义的级别</p> <p>"source": string, 事件来源, 可选。小于 80 字符。</p> <p>"des": string, 事件描述, 小于 160 字符。</p> <p>"sarg": string, 事件参数, < 240 字节 argtype <0 时, 直接 utf8 的 json 字符串。 argtype >0 时, 表示二进制数据需 base64 编码。</p> <p>"cflag": int, 事件处理标志, 0: 未处理; 1: 已处理。提交时无需此字段。</p>
--	---

例子:

请求 1 从主站读取

```
{
  "request": "rdb_soequery",
  "seqno": 2021,
  "time_begin": "2020/2/1",
  "time_end": "2020/3/1",
  "top": 4
}
```

应答 1

```
{
  "response": "rdb_soequery",
  "seqno": 2021,
  "status": 0,
```



```

"message": "OK",
"end": 0,
"vals": [
  {
    "time": "2020-02-23 10:05:10.000",
    "ukey": 0,
    "type": 0,
    "argtype": -1,
    "arglen": 27,
    "level": 0,
    "source": "rdbapidemo",
    "des": "",
    "sarg": "No1:2020- 2-23 10: 5:10. 0",
    "cflag": 1
  },
  {
    "time": "2020-02-23 10:05:10.000",
    "ukey": 1,
    "type": 0,
    "argtype": -1,
    "arglen": 27,
    "level": 0,
    "source": "rdbapidemo",
    "des": "",
    "sarg": "No1:2020- 2-23 10: 5:10. 0",
    "cflag": 1
  },
  {
    "time": "2020-02-23 10:08:10.000",
    "ukey": 0,
    "type": 0,
    "argtype": -1,
    "arglen": 27,
    "level": 0,
    "source": "rdbapidemo",
    "des": "",
    "sarg": "No1:2020- 2-23 10: 8:10. 0",
    "cflag": 0
  },
  {
    "time": "2020-02-23 10:08:10.000",
    "ukey": 1,

```

```

        "type": 0,
        "argtype": 101,
        "arglen": 8,
        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "ZLEyrwMAAAA=",
        "cflag": 0
    }
]
}

```

请求 2 从子站查询

```

{
    "request": "sub_soequery",
    "seqno": 2022,
    "station": "sub1",
    "time_begin": "2020/2/1",
    "time_end": "2020/3/1",
    "top": 4
}

```

应答 2

```

{
    "response": "sub_soequery",
    "seqno": 2022,
    "station": "sub1",
    "status": 0,
    "message": "OK",
    "end": 0,
    "vals": [
        {
            "time": "2020-02-23 10:05:10.000",
            "ukey": 0,
            "type": 0,
            "argtype": -1,
            "arglen": 27,
            "level": 0,
            "source": "rdbapidemo",
            "des": "",
            "sarg": "No1:2020- 2-23 10: 5:10. 0",
            "cflag": 1
        }
    ]
}

```

```

    },
    {
        "time": "2020-02-23 10:05:10.000",
        "ukey": 1,
        "type": 0,
        "argtype": -1,
        "arglen": 27,
        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "No1:2020- 2-23 10: 5:10. 0",
        "cflag": 1
    },
    {
        "time": "2020-02-23 10:08:10.000",
        "ukey": 0,
        "type": 0,
        "argtype": -1,
        "arglen": 27,
        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "No1:2020- 2-23 10: 8:10. 0",
        "cflag": 0
    },
    {
        "time": "2020-02-23 10:08:10.000",
        "ukey": 1,
        "type": 0,
        "argtype": 101,
        "arglen": 8,
        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "ZLEyrwMAAAA=",
        "cflag": 0
    }
}
]
}

```

7.2 事件写入 rdb_soewrite

一般用于数据网关将 SOE 事件提交到实时库。该命令写入直接连接的实时库，不能经过主站代理。

每次可以写入多个 SOE 事件。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_soewrite"
seqno	int	>0, 客户端填写, 服务端原样返回
vals	object array	<p>SOE 记录集, JSON 对象数组, 每个标签属性的字段定义:</p> <p>"time" : 时标, string 或者 number, 参见 2.3 时标。</p> <p>"ukey" :int 可选, 该字段用于当时标重复时作为辅助 KEY 使用。写入时无需此字段 0;</p> <p>"type": int 可选, 默认 0, 应用层定义的事件类型。</p> <p>"argtype": int, 可选, 默认 0: <0 为基于字符串的格式, >0 为基于 2 进制的格式。 具体有用户定义, 用于客户端解析参数。</p> <p>"arglen": int , 参数长度。可选, 默认 0 表示没有参数。</p> <p>"level": int, 事件级别, 可选, 默认 0, 应用层定义的级别</p> <p>"source": string, 事件来源, 可选。小于 80 字符。</p> <p>"des": string, 事件描述, 小于 160 字符。</p> <p>"sarg": string, 事件参数, < 240 字节 argtype <0 时, 直接 utf8 的 json 字符串。 argtype >0 时, 表示二进制数据需 base64 编码。</p> <p>"cflag":int, 事件处理标志, 0: 未处理; 1: 已处理。提交时无需此字段。</p>

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_soewrite"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
write_recs	int	成功写入的 SOE 记录数。

例子

请求 1 写入

```
{
  "request": "rdb_soewrite",
  "seqno": 2031,
  "vals": [
    {
      "time": "2020-3-2 18:32:41.200",
      "source": "test.f32",
      "des": "test.f32 write soe",
      "argtype": -1,
      "arglen": 4,
      "sarg": "12.5"
    },
    {
      "time": "2020-3-2 18:32:41.200",
      "source": "test.i32",
      "des": "test.i32 write soe",
      "argtype": -1,
      "arglen": 3,
      "sarg": "125"
    }
  ]
}
```

写入两条相同时标的 SOE, 系统会自动分配 ukey。

应答 1

```
{
  "response": "rdb_soewrite",
  "seqno": 2031,
  "status": 0,
```

```

    "message": "OK",
    "write_recs": 2
}

```

7.3 事件更新 rdb_soeupdate

事件更新一般用于处理事件。该命令更新直接连接的实时库，不能经过主站代理，每次可更新多个 SOE 事件。

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_soeupdate"
seqno	int	>0, 客户端填写, 服务端原样返回
vals	object array	<p>SOE 记录集, JSON 对象数组, 每个标签属性的字段定义:</p> <p>"time" : string 或者 number, 参见 2.3 时标。</p> <p>"ukey" : 该字段用于当时标重复时作为辅助 KEY 使用。</p> <p>"type": int 可选, 默认 0, 应用层定义的事件类型。</p> <p>"argtype": int, 可选, 默认 0: <0 为基于字符串的格式, >0 为基于 2 进制的格式。 具体有用户定义, 用于客户端解析参数。</p> <p>"arglen": int , 参数长度。可选, 默认 0 表示没有参数。</p> <p>"level": int, 事件级别, 可选, 默认 0, 应用层定义的级别</p> <p>"source": string, 事件来源, 可选。小于 80 字符。</p> <p>"des": string, 事件描述, 小于 160 字符。</p> <p>"sarg": string, 事件参数, < 240 字节 argtype <0 时, 直接 utf8 的 json 字符串。 argtype >0 时, 表示二进制数据需 base64 编码。</p> <p>"cflag": int, 事件处理标志, 0: 未处理; 1: 已处理。</p>

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_soeupdate"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object array	<p>SOE 记录集, JSON 对象数组, 每个标签属性的字段定义:</p> <p>"time" : 时标, string 或者 number, 参见 2.3 时标。</p> <p>"ukey" : 该字段用于当时标重复时作为辅助 KEY 使用。</p> <p>"type": int 可选, 默认 0, 应用层定义的事件类型。</p> <p>"argtype": int, 可选, 默认 0: <0 为基于字符串的格式, >0 为基于 2 进制的格式。 具体有用户定义, 用于客户端解析参数。</p> <p>"arglen": int , 参数长度。可选, 默认 0 表示没有参数。</p> <p>"level": int, 事件级别, 可选, 默认 0, 应用层定义的级别</p> <p>"source": string, 事件来源, 可选。小于 80 字符。</p> <p>"des": string, 事件描述, 小于 160 字符。</p> <p>"sarg": string, 事件参数, < 240 字节 argtype <0 时, 直接 utf8 的 json 字符串。 argtype >0 时, 表示二进制数据需 base64 编码。</p> <p>"cstatus": int , 更新结果; 0: 成功更新; 1: 不存在或更新失败</p> <p>"cflag": int, 事件处理标志, 0: 未处理; 1: 已处理。</p>

例子:

请求 1 设置事件已处理, 把事件都回来, 设置 cflag=1;

```

{
  "request": "rdb_soeupdate",
  "seqno": 2032,
  "vals": [
    {
      "time": "2020-3-2 18:32:41.200",
      "ukey": 0,
      "source": "test.f32",
      "des": "test.f32 write soe",
      "argtype": -1,
      "arglen": 4,
      "sarg": "12.5",
      "cflag": 1
    },
    {
      "time": "2020-3-2 18:32:41.200",
      "ukey": 1,
      "source": "test.i32",
      "des": "test.i32 write soe",
      "argtype": -1,
      "arglen": 3,
      "sarg": "125",
      "cflag": 1
    }
  ]
}

```

应答 1

```

{
  "response": "rdb_soeupdate",
  "seqno": 2032,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "time": "2020-03-02 18:32:41.200",
      "ukey": 0,
      "type": 0,
      "argtype": -1,
      "arglen": 4,
      "level": 0,
      "source": "test.f32",

```



```

        "des": "test.f32 write soe",
        "sarg": "12.5",
        "cstatus": 0,
        "cflag": 1
    },
    {
        "time": "2020-03-02 18:32:41.200",
        "ukey": 1,
        "type": 0,
        "argtype": -1,
        "arglen": 3,
        "level": 0,
        "source": "test.i32",
        "des": "test.i32 write soe",
        "sarg": "125",
        "cstatus": 0,
        "cflag": 1
    }
]
}

```

注意：使用该命令时，除了 **time** 和 **ukey** 字段(**time** 和 **ukey** 组合作为 **KEY**)，其他都是覆盖更新。更新时使用订阅回来的事件或者使用 **rdb_soquery** 查询出来的事件上更改需要更改的字段，无需更改的字段要保持不变。

7.4 事件订阅 rdb_soessc

订阅 **SOE** 事件，当服务端有满足订阅条件的事件时会主动推送给客户端。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_soessc"
seqno	int	>0，客户端填写，服务端原样返回
station	string	子站名(连接中心站才有效)，无此字段表示从当前连接站订阅
flag	int	订阅方式： 0:取消订阅； 1:订阅最新； 2:订阅更新的 SOE (被处理后的 SOE)； 3:订阅最新和更新后的 SOE

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_modfypswd"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。

例子

请求 1

```
{
    "request": "rdb_soessc",
    "seqno": 2040,
    "flag": 1
}
```

应答 1

```
{
    "response": "rdb_soessc",
    "seqno": 2040,
    "status": 0,
    "message": "OK"
}
```

7.5 事件推送 put_soe

订阅成功后, 如果有事件满足条件, 服务端会主动推送。

推送事件格式:

KEY	VAL 类型	说明
response	string	"put_soe"
seqno	int	0, 始终为 0
status	int	0, 始终为 0
message	string	始终为"OK"
vals	object array	SOE 记录集, JSON 对象数组, 每个标签属性的字段定义:

	<p>"time" : 时标, string 或者 number, 参见 2.3 时标。</p> <p>"ukey" : 该字段用于当时标重复时作为辅助 KEY 使用。</p> <p>"type": int 可选, 默认 0, 应用层定义的事件类型。</p> <p>"argtype": int, 可选, 默认 0: <0 为基于字符串的格式, >0 为基于 2 进制的格式。 具体有用户定义, 用于客户端解析参数。</p> <p>"arglen": int , 参数长度。可选, 默认 0 表示没有参数。</p> <p>"level": int, 事件级别, 可选, 默认 0, 应用层定义的级别</p> <p>"source": string, 事件来源, 可选。小于 80 字符。</p> <p>"des": string, 事件描述, 小于 160 字符。</p> <p>"sarg": string, 事件参数, < 240 字节 argtype <0 时, 直接 utf8 的 json 字符串。 argtype >0 时, 表示二进制数据需 base64 编码。</p> <p>"cflag": int, 事件处理标志, 0: 未处理; 1: 已处理。</p>
--	---

例子:

```
{
  "response": "put_soe",
  "seqno": 0,
  "status": 0,
  "message": "OK",
  "vals": [
    {
      "time": "2020-03-13 10:08:20.000",
      "ukey": 0,
      "type": 0,
      "argtype": -1,
      "arglen": 27,
```

```

        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "No1:2020- 3-13 10: 8:20. 0",
        "cflag": 0
    },
    {
        "time": "2020-03-13 10:08:20.000",
        "ukey": 1,
        "type": 0,
        "argtype": 101,
        "arglen": 8,
        "level": 0,
        "source": "rdbapidemo",
        "des": "",
        "sarg": "yC4tsAMAAAA=",
        "cflag": 0
    }
]
}

```

7.6 删除 SOE 数据 rdb_soedeleter

用于删除指定时间段的 SOE 数据。(2023.4 版，内部版本 5105 开始支持本命令)

请求字段说明

KEY	VAL 类型	说明
request	string	"rdb_soedeleter"
seqno	int	>0，客户端填写，服务端原样返回
time_begin	string/number	开始时间(含)，string 或者 number，参见 2.3 时标
ukey_begin	number	相同开始时间的序号，可选，默认 0
time_end	string/number	结束时间(不含)，string 或者 number，参见 2.3 时标；不提供此字段或者填写-1 表示从开始时标之后的数据全部删除。
ukey_end	Number	相同结束时间的序号，可选。默认 0

应答字段说明

KEY	VAL 类型	说明
-----	--------	----

response	string	"rdb_soedelete"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
records	Number	删除的记录数

例子 1, 删除一个月 SOE 数据 。

请求:

```
{
  "request": "rdb_soedelete",
  "seqno": 3038,
  "time_begin": "2023-1-1T0:0:0.000+08:00",
  "ukey_begin": 0,
  "time_end" : "2023-2-1T0:0:0.000+08:00",
  "ukey_end": 0
}
```

应答:

```
{
  "response": "rdb_soedelete",
  "seqno": 3038,
  "status": 0,
  "message": "OK",
  "records": 231
}
```

8 系统监控和维护命令详解

本章描述获取系统运行信息的命令。

8.1 读取在线子站列表

从中心站中读取接入的子站列表。

此命令 2021.2 版新增。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_liststation"
seqno	int	>0, 客户端填写, 服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_liststation"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object	在线子站信息对象数组, 具体格式参见例子。

例子：

请求：

```
{
    "request": "rdb_liststation",
    "seqno": 8001
}
```

应答：

```
{
    "response": "rdb_liststation",
    "seqno": 8001,
    "vals": [{
        "stationid": "sub1.master",
        "ucid": 1001,
        "peer": "127.0.0.1:50179",
        "time_login": "2021/3/1 14:37:11"
    }, {
```

```

        "stationid": "sub1.slave",
        "ucid": 1002,
        "peer": "127.0.0.1:50180",
        "time_login": "2021/3/1 14:37:12"
    }
]
}

```

8.2 读取实时库运行信息

读取当前连接的实时库中接入本实时库的数据网关信息。或者从中心站读取子站的运行信息。

此命令 2021.2 版新增。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_getruninfo"
seqno	int	>0, 客户端填写, 服务端原样返回
stationid	string	子站 ID, 无此字段表示读取当前连接的库, 否则表示读取子站的信息。 stationid 用 "子站名.master" 表示子站主站, "子站 id.slave" 表示子站从站。

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_getruninfo"
seqno	int	>0, 客户端填写, 服务端原样返回
stationid	string	有则原样返回, 总是有 ".master" 或者 ".slave" 后缀
status	int	0 表示成功, 其余为错误码。这是总状态, 如果不为 0, 就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object	子站运行信息, 具体参见例子。

例子：

请求：

```

{
    "request": "rdb_getruninfo",
    "seqno": 8002,

```

```

    "stationid": "sub1.master"
}

```

应答:

```

{
  "response": "rdb_getruninfo",
  "seqno": 8002,
  "stationid": "sub1.master",
  "status": 0,
  "vals": {
    "gateways": [{
      "peerurl": "127.0.0.1:58179",
      "ucid": 1001,
      "login_time": "2021/2/18 14:44:16",
      "update_time": "2021/2/18 14:48:32",
      "runinfo": {
        "dacname": "localsimu",
        "ha_type": "HA_NONE",
        "master_db": {
          "url": "ws://127.0.0.1:921",
          "status": "ONLINE"
        },
        "devices": [{
          "name": "simu",
          "tags": 1024,
          "status": "GOOD",
          "ha_status": "WORK"
        }, {
          "name": "opcua1",
          "tags": 121,
          "status": "E_DEVICE",
          "ha_status": "WORK"
        }
      ]
    }, {
      "peerurl": "127.0.0.1:58179",
      "ucid": 1002,
      "login_time": "2021/2/18 14:44:16",
      "update_time": "2021/2/18 14:48:32",
      "runinfo": {
        "dacname": "localsimu",
        "ha_type": "HA_NONE",
        "master_db": {
          "url": "ws://127.0.0.1:921",
          "status": "ONLINE"
        },
        "devices": [{
          "name": "simu",
          "tags": 1024,
          "status": "GOOD",
          "ha_status": "WORK"
        }, {
          "name": "opcua1",
          "tags": 121,
          "status": "E_DEVICE",
          "ha_status": "WORK"
        }
      ]
    }
  ]
}

```

注: **vals** 中 **gateways** 是数据网关对象数组, 每个数据网关对象里又有设备数组。

8.3 读取实时库连接会话列表

读取当前连接的实时库的客户端连接会话列表，或者从中心站读取子站的客户端连接会话列表。

此命令 2021.3 版新增。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_listsession"
seqno	int	>0，客户端填写，服务端原样返回
stationid	string	子站 ID，无此字段表示读取当前连接的库，否则表示读取子站的信息。 stationid 用 "子站名.master" 表示子站主站， "子站 id.slave" 表示子站从站。

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_listsession"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。这是总状态，如果不为 0，就没有 vals 字段。
message	string	当 status !=0 时对错误的具体描述。
vals	object	在线子站信息对象数组，具体格式参见例子。

例子：

请求：

```
{
  "request": "rdb_listsession",
  "seqno": 8003
}
```

应答：

```
{
  "response": "rdb_listsession",
  "seqno": 8003,
  "status": 0,
  "message": "OK",
  "vals": [{
    "ucid": 1001,
    "protocol": "TCP",
    "peer": "127.0.0.1:50179",
```

```

        "user": "",
        "status": 0,
        "login_time": "2021/3/13 15:18:52"
    }, {
        "ucid": 1008,
        "protocol": "WS",
        "peer": "127.0.0.1:50207",
        "user": "admin",
        "status": 1,
        "login_time": "2021/3/13 15:19:9"
    }
]
}

```

8.4 断开实时库连接会话连接

系统管理员用此命令断开客户端连接，可以按照会话连接 **ID** 和用户断开。可以断开已登录和未登录的连接，一般用于减少了某客户的连接数配置后，断开该用户所有连接触发该用户的客户端重连。此命令 **2021.6** 版新增。

请求格式：

KEY	VAL 类型	说明
request	string	"rdb_closesession"
seqno	int	>0，客户端填写，服务端原样返回
sessionid	number	会话连接 ID
user	string	用户账号，断开该用户的所有连接。

注：不同时提供 **user** 和 **sessionid** 字段，如果同时提供只采用 **sessionid**。

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_listsession"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。
message	string	当 status !=0 时对错误的具体描述。
sessionid 或 user	number 或 string	

例子 1：

请求：

```
{ "request": "rdb_closesession", "seqno": 8024, "sessionid": 1024 }
```

应答:

```
{
  "response": "rdb_closesession",
  "seqno": 8024,
  "sessionid": 1024,
  "status": 0,
  "message": "OK"
}
```

例子 2:

请求:

```
{"request": "rdb_closesession", "seqno": 8025, "user": "opt1"}
```

应答:

```
{
  "response": "rdb_closesession",
  "seqno": 8025,
  "user": "opt1",
  "status": 0,
  "message": "OK"
}
```

8.5 重建指定标签历史数据索引

需要系统管理员权限，因磁盘故障或者断电故障导致标签数据页面或索引页面损坏，且自动修改失败时，在 web 版的命令行控制台发送此命令剔除坏页面并修复改标签索引，此命令 2025.3(inver 5122)版新增。此命令将扫描任务加入后台任务列表后立即返回。

请求格式:

KEY	VAL 类型	说明
request	string	"rdb_rebuildtagindex"
seqno	int	>0，客户端填写，服务端原样返回
tag	string	标签名

应答字段说明

KEY	VAL 类型	说明
response	string	"rdb_rebuildtagindex"
seqno	int	>0，客户端填写，服务端原样返回
status	int	0 表示成功，其余为错误码。
message	string	对 status 的具体描述

例子 1:

请求:

```
{"request": "rdb_rebuilddtagindex", "seqno": 2022, "tag": "d0.f01.pv"}
```

应答:

```
{
  "response": "rdb_rebuilddtagindex",
  "seqno": 2022,
  "status": 0,
  "message": "start d0.f01.pv rebuilding"
}
```

8.6 内存池信息查询

Rdbsrv 里面的所有内存均是有这个内存池分配, 通过 `mem_info` 命令查询内存使用情况。此命令 2025.3(inver 5122)版新增。

请求格式:

KEY	VAL 类型	说明
request	string	"sys_meminfo"
seqno	int	>0, 客户端填写, 服务端原样返回

应答字段说明

KEY	VAL 类型	说明
response	string	"sys_meminfo"
seqno	int	>0, 客户端填写, 服务端原样返回
status	int	0 表示成功, 其余为错误码。
vals	object	返回的内存池信息

例子 1:

请求:

```
{"request": "sys_meminfo", "seqno": 2022}
```

应答:

```
{
  "response": "sys_meminfo",
  "seqno": 2022,
  "status": 0,
  "vals": [
    {
      "heap_objects": [

```

```

        "numobjs": 8191,
        "freeobjs": 8093,
        "sizearea": 524288,
        "sizeobj": 64
    }
],
"numheaps": 1,
"numfreeblks": 8093
},
{
    "numLargeMemorys": 0
},
{
    "blockSize": 16,
    "blockPerHeap": 65536,
    "numHeaps": 2,
    "FreeBlocks": 47004
},
{
    "blockSize": 32,
    "blockPerHeap": 32768,
    "numHeaps": 2,
    "FreeBlocks": 23498
},
{
    "blockSize": 64,
    "blockPerHeap": 16384,
    "numHeaps": 1,
    "FreeBlocks": 455
},
{
    "blockSize": 96,
    "blockPerHeap": 10922,
    "numHeaps": 2,
    "FreeBlocks": 10982
},
{
    "blockSize": 128,
    "blockPerHeap": 8192,
    "numHeaps": 2,
    "FreeBlocks": 6520
},
{
    "blockSize": 256,
    "blockPerHeap": 4096,
    "numHeaps": 1,
    "FreeBlocks": 113
},
{
    "blockSize": 384,

```

```

        "blockPerHeap": 2730,
        "numHeaps": 5,
        "FreeBlocks": 715
    },
    {
        "blockSize": 512,
        "blockPerHeap": 2048,
        "numHeaps": 1,
        "FreeBlocks": 62
    },
    {
        "blockSize": 640,
        "blockPerHeap": 1638,
        "numHeaps": 1,
        "FreeBlocks": 48
    },
    {
        "blockSize": 800,
        "blockPerHeap": 1310,
        "numHeaps": 1,
        "FreeBlocks": 40
    },
    {
        "blockSize": 1024,
        "blockPerHeap": 1024,
        "numHeaps": 1,
        "FreeBlocks": 31
    },
    {
        "blockSize": 1424,
        "blockPerHeap": 736,
        "numHeaps": 1,
        "FreeBlocks": 23
    },
    {
        "blockSize": 2048,
        "blockPerHeap": 4096,
        "numHeaps": 1,
        "FreeBlocks": 13
    },
    {
        "blockSize": 3072,
        "blockPerHeap": 2730,
        "numHeaps": 1,
        "FreeBlocks": 15
    },
    {
        "blockSize": 5120,
        "blockPerHeap": 1638,
        "numHeaps": 1,

```

```

    "FreeBlocks": 8
  },
  {
    "blockSize": 8192,
    "blockPerHeap": 1024,
    "numHeaps": 2,
    "FreeBlocks": 1024
  },
  {
    "blockSize": 12288,
    "blockPerHeap": 682,
    "numHeaps": 1,
    "FreeBlocks": 15
  },
  {
    "blockSize": 16384,
    "blockPerHeap": 512,
    "numHeaps": 1,
    "FreeBlocks": 14
  },
  {
    "blockSize": 20480,
    "blockPerHeap": 409,
    "numHeaps": 24,
    "FreeBlocks": 19
  },
  {
    "blockSize": 24576,
    "blockPerHeap": 341,
    "numHeaps": 1,
    "FreeBlocks": 16
  },
  {
    "blockSize": 32768,
    "blockPerHeap": 256,
    "numHeaps": 5,
    "FreeBlocks": 155
  },
  {
    "blockSize": 49152,
    "blockPerHeap": 170,
    "numHeaps": 1,
    "FreeBlocks": 16
  },
  {
    "blockSize": 65536,
    "blockPerHeap": 128,
    "numHeaps": 1,
    "FreeBlocks": 16
  },

```

```

{
    "blockSize": 131072,
    "blockPerHeap": 128,
    "numHeaps": 1,
    "FreeBlocks": 15
},
{
    "blockSize": 262144,
    "blockPerHeap": 64,
    "numHeaps": 1,
    "FreeBlocks": 10
},
{
    "blockSize": 409600,
    "blockPerHeap": 40,
    "numHeaps": 1,
    "FreeBlocks": 14
},
{
    "blockSize": 655360,
    "blockPerHeap": 25,
    "numHeaps": 1,
    "FreeBlocks": 8
},
{
    "blockSize": 1048576,
    "blockPerHeap": 2,
    "numHeaps": 0,
    "FreeBlocks": 0
},
{
    "blockSize": 2097152,
    "blockPerHeap": 2,
    "numHeaps": 1,
    "FreeBlocks": 2
},
{
    "blockSize": 4194304,
    "blockPerHeap": 2,
    "numHeaps": 2,
    "FreeBlocks": 1
},
{
    "blockSize": 8388608,
    "blockPerHeap": 2,
    "numHeaps": 0,
    "FreeBlocks": 0
},
{
    "netio_inbufmsgs": 0,

```



```
        "netio_outbufmsgs": 0,  
        "netio_syncinbufmsgs": 0,  
        "netio_globaloutbufheaps": 1  
    }  
]  
}
```

9 协议实现

本协议可以使用 C/C++, C#, java, javascript 实现。

9.1 websocket 的 PING/PONG 消息

传输层 websocket 通道上的 PING/PONG 消息作为 **keepalive** 机制不是必须的，因为 **tcp** 层已经有 **keepalive** 机制。

在 **rdb2021.9** 之前版本没有对 PING/PONG 消息做处理(是错误消息从而将客户端断开)，从 **rdb2021.9** 开始对 PING 消息返回 PONG 消息，对于 PONG 消息按照规范滤掉。

chrome, **firefox**, **Safari** 以及 **ipad**, **iphone**, **Android** 内置浏览器均不会发送 PING / PONG 消息，因此 JS 编码无限制。

j2ee 中自带的 **websocket** 客户端也不会发送 PING/PONG 消息。如果使用第三方 **org** 的客户端，注意要关闭定时发送 PING/PONG 消息。

C#自带的 **ClientWebSock** 默认会 30 秒发送一次 PONG 消息。连接 **rdb2021.9** 之前版本需要设置 **ClientWebSock.Options.KeepAliveInterval=0** 关闭 PING/PONG 消息定时发送功能。

10 更新记录

- 1) 2021.9 版, 增加后台 **soe** 事件扩展属性事件级别。参见 **rdb_tagimport_ext** 和 **rdb_tagget_ext** 命令, 需要 **rdbsrv2021.9** 版本支持。增加 **websocket** 的 **PING/PONG** 消息
- 2) 2022.7SP1 版, 增加 **rdb_tagexport** 命令。
- 3) 2023.3 版增加 **ISO** 时标和 **javascript** 数字型 **timestamp** 时标, 参见 2.3 时标描述。
- 4) 2023.4 版(内部版本 5105)增加数据和 **SOE** 删除接口, 数据无压缩插入接口, 工作模式读取和设置接口
- 5) 2023.6 版(内部版本 5107)增加会话连接删除接口。
- 6) 2023.7 版(内部版本 5108), 增加全 **RFC8259JSON** 转义完全支持, 修改 1.4**JSON** 规范。
- 7) 2023.12 版(内部版本 5114), 增加 **rdb_valquery** 命令 **lflag** 参数枚举值定义(区间最大最小值), 增加 **rdb_plotdata** 接口可选结束时间参数。
- 8) 2024.9 版(内部版本 5119), 增加订阅推送相关协议。
- 9) 2025.3 版(内部版本 5122), 增加标签属性中基本数据类型的更改。增加 **rdb_valquery** 命令 **lflag** 参数枚举值定义(冷备模式读取)。增加重建标签历史数据索引命令, 增加内存池信息查询。
- 10) 2025.4 版(内部版本 5123), 增加增量统计和积分累计量统计。