GDB 实时数据库系统接口文档

1.简介

GDB 数据库接口使用的是标准的 restful 格式，返回的数据包括三个字段，分别是 code，message，data，所有的请求方式均为 POST，如果 gdb 使用的是授权模式，则需要在请求的头部加上 Authorization 字段，字段的值即为登陆成功获取的 token 字段

字段名称	字段类型	字段含义
code	int	状态码，200 为成功，500 为失败
message	string	额外的信息，如果 code 为 200 则为空字符串，code 为 500 则为请求失败的原因
data	interface{}	响应的数据，如果请求失败则为空字符串

以下所有的代码均是在 ES6 环境下，使用 NodeJS + axios 测试完成的，至于更多的细节可以查看 github：https://github.com/JustKeepSilence/gdbUI。

注意事项：

Go 中的 interface{}类型可以理解成其他语言中的 object 类型。

2. Examples

整个 GDB 的接口文档分为 Page，Group，Item，Data 四个部分，分别定义了和页面，分组，item 以及数据相关的操作。

2.1 Page

(1) /page/ userLogin

请求描述：用户登陆

请求字段：

字段名称	字段描述	字段类型	是否必须
userName	用户名称	string	是
passWord	用户密码	string	是

请求示例：

axios.post("/page/userLogin",{userName:"admin",passWord:"685a6b21dc732a9702a96e6731811ec9"})

{"code":200,"message":"","data":{"token":"bc947ca95872df7993fb277072eaa12d"}}

注意事项：

登陆时候的密码为 md5(`${passWord}@seu`)

(2) /page/userLogOut

请求描述：用户退出登陆

请求字段：

字段名称	字段描述	字段类型	是否必须
userName	用户名称	string	是

请求示例：

axios.post("/page/userLogOut",{userName: "admin"})

{"code":200,"message":"","data":{"effectedRows":1}}

(3) /page/getUserInfo

请求描述：获取用户信息

请求字段：

字段名称	字段描述	字段类型	是否必须
name	用户名称	string	是

请求示例：

axios.post("/page/getUserInfo", {"name": "admin"})

{"code":200,"message":"","data":{"userName":"admin","role":["super_user"]}}

(4) /page/getUsers

请求描述：获取 gdb 中所有的用户

请求字段：无

请求示例：

axios.post("/page/getUsers")

{"code":200,"message":"","data":{"userInfos":[{"id":"1","role":"super_user","userName":"admin"}]}}

(5) /page/addUsers

请求描述：向 gdb 中增加用户

请求字段：

字段名称	字段描述	字段类型	是否必须
name	用户名称	string	是
role	用户角色	string	是
passWord	用户密码	string	是

axios.post("/page/addUsers",{"name":"seu","role":"common_user","passWord":"685a6b21dc732a9702a96e6731811ec9"})

{"code":200,"message":"","data":{"effectedRows":1}}

注意事项：用户角色只能是 visitor, super_user, common_user 其中之一，并且用户名不能重复

(6) /page/deleteUsers

请求描述：删除 gdb 中的用户

请求字段：

字段名称	字段描述	字段类型	是否必须
name	用户名称	string	是

请求示例：

axios.post("/page/deleteUsers", {"name":"seu"})

{"code":200,"message":"","data":{"effectedRows":1}}

(7) /page/updateUsers

请求描述：更新用户信息

请求字段：

字段名称	字段描述	字段类型	是否必须
userName	原来的用户名称	int	是
newUserName	新的用户名称	string	是
newPassWord	新密码	string	否
newRole	新用户角色	string	是

请求示例：

axios.post("/page/updateUsers",{"userName":"seu1","newUserName":"seu1","newPassWord":"685a6b21dc732a9702a96e6731811ec9","newRole":"common_user"})

{"code":200,"message":"","data":{"effectedRows":1}}

注意事项：

更新用户信息时如果不传入 newPassWord 字段则意味着不更新密码，不传入 newUserName 则意味着不更新 userName,不传入 newRole 则意味着不更新 role。

(8) /page/getLogs

请求描述：获取 gdb 运行日志

请求字段：

字段名称	字段描述	字段类型	是否必须
level	日志等级	string	是
startTime	开始时间	string	是
endTime	结束时间	string	是
startRow	分页查询开始的 row	int	是
rowCount	分页查询每页的数目	int	是
name	用户名称	string	是

请求示例：

axios.post("/page/getLogs", {"level":"all","startTime":"2021-05-23 14:42:29","endTime":"2021-05-24 14:42:29","startRow":0,"rowCount":10,"name":"admin"})

注意事项：

level 字段代表的是要获取日志的等级，只能是 all(所有等级)，Info,Error 之一。

(9) /page/deleteLogs

请求描述：删除日志

请求字段：

字段名称	字段描述	字段类型	是否必须
id	数据库的 id	string	否
startTime	开始时间	string	否
endTime	结束时间	string	否
userNameCondition	用户名称条件	string	否

请求示例：

axios.post("/page/deleteLogs", {"id":"8"})

{"code":200,"message":"","data":{"effectedRows":1}}

axios.post("/page/deleteLogs", {"startTime":"2021-05-23 15:12:06","endTime":"2021-05-24 15:12:06","userNameCondition":"requestUser='admin'"})

{"code":200,"message":"","data":{"effectedRows":10}}

注意事项：

如果提供了 id，则意味着只会删除这一条数据。如果想批量删除日志，则需要提供 startTime,endTime 以及 userNameCondition 字段，其中 userNameCondition 字段指定了去删除哪些用户记录的日志，如果是 1=1 则意味着删除指定时间条件下的所有用户的日志。对于在非授权模式下记录的日志，其 userName 为空字符串。

(10) /page/getDbInfo

请求描述：获取 gdb 运行时的相关信息

请求字段：无

请求示例：

axios.post("/page/getDbInfo")

{"code":200,"message":"","data":{"info":{"currentTimeStamp":"1621869963","ram":"169.14","speed":"51ms/6137","writtenItems":"6137"}}}

注意事项：返回的字段的含义分别为

currentTimeStamp：最近一次 gdb 数据库更新的 unix 时间戳

ram：当前 gdb 的内存使用，单位为 M

speed：当前时刻 gdb 的写入速率

writtenItems：当前时刻 gdb 写入的 item 个数

(11) /page/ getDbInfoHistory

请求描述：获取 gdb 运行过程中的内存或者写入耗时的历史数据

请求字段：

字段名称	字段描述	字段类型	是否必须
itemName	要获取历史数据的 item 名称	string	是
startTimes	开始时间戳	[]int	是
endTimes	结束时间戳	[]int	是
intervals	取数时间间隔	[]int	是

请求示例：

axios.post("/page/getDbInfoHistory",{"itemName":"speed","startTimes":[1621209000],"endTimes":[1621382400],"intervals":[3600]})

{"code":200,"message":"","data":{"historicalData":{"speed":[["1621209000","1621213881","1621218786","1621223692","1621228582","1621233489","1621238397","1621243288","1621248199","1621253118","1621258008","1621262917","1621267818","1621272727","1621277619","1621282529","1621287427","1621292329","1621297233","1621302117","1621307022","1621311938","1621316850","1621321740","1621326637","1621331533","1621336424","1621341332","1621346240","1621351149","1621356048","1621360950","1621365846","1621370744","1621375634","1621380559"],["81.0773ms","107.1021ms","71.0679ms","167.1601ms","56.0528ms","109.1051ms","72.067ms","35.033ms","60.0555ms","49.047ms","114.1066ms","136.1308ms","98.0942ms","159.1516ms","85.082ms","98.0938ms","56.0524ms","98.0942ms","79.0724ms","116.1117ms","181.1737ms","82.08ms","140.1347ms","91.0851ms","144.1385ms","84.0801ms","146.1406ms","120.113ms","52.0503ms","83.0774ms","155.1494ms","103.097ms","159.1507ms","66.0635ms","101.0969ms","49.0471ms"]]}}}

注意事项：

itemName 只能是 speed 或者 ram 之一，分别代表获取写入速率或者内存使用的历史数据。

2.2 Group

(1) /page/addGroups

请求描述：向 gdb 中添加分组

请求字段：

字段名称	字段描述	字段类型	是否必须
groupInfos	分组信息	[]groupInfo	是

groupInfo 的字段为：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
columnNames	分组中的列名	[]string	是

请求示例：

axios.post("/group/addGroups",{"groupInfos":[{"groupName":"1DCS","columnNames":["description", "unit"]}]})

{"code":200,"message":"","data":{"effectedRows":1}}

注意事项：

1. 添加分组时不需要添加 id,itemName,dataType 列，这三列是系统自带的，如果添加则会添加分组失败。
2. 添加分组时分组的名称不能和 go 语言中的关键字重复。
3. 分组名称是唯一的，不能重复添加，且 calc 分组为自带的，不可添加和删除

(2) /page/deleteGroups

请求描述：删除 gdb 中的分组

请求字段：

字段名称	字段描述	字段类型	是否必须
groupNames	要删除的分组名称	[]string	是

请求示例：

axios.post("/group/deleteGroups", {"groupNames": ["1DCS"]})

{"code":200,"message":"","data":{"effectedRows":1}}

(3) /page/getGroups

请求描述：获取 gdb 中所有已经存在的分组名称

请求字段：无

请求示例：

axios.post("/group/getGroups")

{"code":200,"message":"","data":{"groupNames":["calc","1DCS"]}}

(4) /page/getGroupProperty

请求描述：获取分组的信息

请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	要获取信息的分组名称	string	是
condition	sqlite 条件语句	string	是

请求示例：

axios.post("/group/getGroupProperty", {"groupName":"1DCS", "condition": "1=1"})

{"code":200,"message":"","data":{"itemCount":"6387","itemColumnNames":["itemName","groupName","dataType","description","unit","source"]}}

返回字段的含义分别为：

itemCount：给定 condition 条件下 groupName 中的 item 个数

itemColumnNames：group 中的列

注意事项：

condition 字段从数据库中筛选 itemCount 的数目，可以用作 table 分页时的筛选条件。

(5) /page/ updateGroupNames

请求描述：更新 group 的名称

请求字段：

字段名称	字段描述	字段类型	是否必须
infos	分组名称信息	[]UpdatedGroupNamesInfo	是

UpdatedGroupNamesInfo 字段为：

字段名称	字段描述	字段类型	是否必须
oldGroupName	原来的分组名称	string	是
newGroupName	新的分组名称	string	是

请求示例：

axios.post("/group/updateGroupNames", {"infos": [{"oldGroupName": "4DCS", "newGroupName": "5DCS"}]})

{"code":200,"message":"","data":{"effectedRows":1}}

注意事项：

在当前 v1.0.5 版本不要使用这个接口，因为更新组名不会迁移对应的历史数据。

(6) /page/updateGroupColumnNames

请求描述：更新 group 中的列名

请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
oldColumnNames	原来的列名	[]string	是
newColumnNames	新的列名	[]string	是

axios.post("/group/updateGroupColumnNames",{"groupName":"1DCS","newColumnNames": ["unit1"], "oldColumnNames": ["unit"]})

{"code":200,"message":"","data":{"effectedCols":1}}

注意事项：

系统自带的 id,itemName,dataType 三列不可更改和删除。

(7) /page/ deleteGroupColumns

请求描述：删除 group 中的列

请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
columnNames	要删除的列名	[]string	是

请求示例：

axios.post("/group/deleteGroupColumns", {"groupName": "1DCS", "columnNames": ["unit"]})

{"code":200,"message":"","data":{"effectedCols":1}}

(8) /page/addGroupColumns
请求描述：向 group 中增加列
请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
columnNames	要增加的列名	[]string	是
defaultValues	列的默认值	[]string	是

请求示例：

```
axios.post("/group/addGroupColumns",{"groupName":"5DCS","columnNames":["unit","description"],"defaultValues":["",""]})
{"code":200,"message":"","data":{"effectedCols":2}}
```

2.3 Item

(1) /item/addItems
请求描述：向 gdb 分组中添加 item
请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
itemValues	要增加的信息	[]map[string]string	是

请求示例：

```
axios.post("/item/addItems",{"groupName":"5DCS","itemValues":[{"itemName":"item1","dataType":"float64","unit":"","description":""}]})
{"code":200,"message":"","data":{"effectedRows":1}}
```

注意事项：
dataType 字段的值只能是 int64,float64,bool,string 四者之一
(2) /item/deleteItems
请求描述：删除 gdb 中的 item
请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
condition	删除的条件语句	string	是

请求示例：

```
axios.post("/item/deleteItems", {"groupName":"5DCS","condition":"itemName='item1'"})
{"code":200,"message":"","data":{"effectedRows":1}}
```

(3) /item/getItemsWithCount
请求描述：获取 gdb 分组中的 item
请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
condition	筛选 item 的条件语句	string	是
columnNames	要获取的列名	string	是

startRow	分页查询开始的row	int	是
rowCount	分页查询每页的数据数目	int	否

请求示例：

axois.post("/item/getItemsWithCount",{"groupName":"calc","columnNames":"*","condition":"itemName like '%%'","startRow":0,"rowCount":10})

{"code":200,"message":"","data":{"itemCount":1,"itemValues":[{"dataType":"float64","description":"","id":"1","itemName":"item1"}]}}

注意事项：

如果请求中的 startRow 字段为-1,则意味着查询指定 group 和 condition,columnNames 下的所有 item 的信息数据。

如果 columnNames 为*则意味着返回所有列的信息。

(4) /item/updateItems

请求描述：更新 group 中的 item 信息

请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
condition	条件语句	string	是
clause	更新的语句	string	是

请求示例：

axios.post("/item/updateItems",{"groupName": "1DCS", "condition": "id=1", "clause": "description=' ',unit='℃1'"})

{"code":200,"message":"","data":{"effectedRows":1}}

注意事项：

不能更新 itemName 属性

(5) /item/checkItems

请求描述：检查指定的 item 是否存在

请求字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
itemNames	要检查的 item	[]string	是

请求示例：

axios.post("/item/checkItems",{"groupName":"1DCS","itemNames":["NMJL.UNIT2.20ACS:MAG50AN001SV_MA"]})

{"code":500,"message":"itemName:NMJL.UNIT2.20ACS:MAG50AN001SV_MA not existed","data":""}

(6) /item/cleanGroupItems

请求描述：删除指定 group 中的所有 item

请求字段：

字段名称	字段描述	字段类型	是否必须
groupNames	分组名称	[]string	是

请求示例：

axios.post("/item/cleanGroupItems",{"groupNames": ["1DCS"]})

{"code":200,"message":"","data":{"effectedRows":1}}
注意事项：
删除 group 中的 item 并不会删除其历史数据。

2.4 Data

(1) /data/batchWrite
请求描述：向 gdb 中写写入实时数据
请求字段：

字段名称	字段描述	字段类型	是否必须
itemValues	Item 实时值信息	[]itemValue	是

itemValue 字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
itemName	item 名称	string	是
value	item 实时值	interface{}	是

请求示例：

```
axios.post("/data/batchWrite", {"itemValues": [{"itemName": "x", "value": 1.0, "groupName": "5DCS"}, {"itemName": "y", "value": 2.0, "groupName": "5DCS"}]})
```

{"code":200,"message":"","data":{"effectedRows":2}}
注意事项：

1. 在对应的分组中必须存在 itemName
2. value 的值必须与 itemName 的 dataType 相对应

(2) /data/ batchWriteHistoricalData
请求描述：批量写入历史数据
请求字段：

字段名称	字段描述	字段类型	是否必须
historicalItemValues	Item 历史值信息	[]HistoricalItemValue	是

HistoricalItemValue 字段：

字段名称	字段描述	字段类型	是否必须
groupName	分组名称	string	是
itemName	item 名称	string	是
values	item 历史值	[]interface{}	是
timeStamps	Item 历史值时间戳	[]int	是

请求示例：

```
'use strict';
const axios = require('axios')
const ip = "192.168.0.199:8082"
const count = 3600 // one hour
const now = new Date(2021,4,24,19,44,0)
const st = now.getTime() / 1000 + 8 * 3600
let xData = []
let yData = []
```

```
let ts = []
for (var i = 0; i < count; i++) {
    const x = Math.floor(Math.random() * count)
    const y = 2 * x
    xData.push(x)
    yData.push(y)
    ts.push(st + i)
}
axios.post(`http://${ip}/data/batchWriteHistoricalData`, { "historicalItemValues": [{ "groupName": "5DCS", "itemName": "x", "values": xData, "timeStamps": ts }, { "groupName": "5DCS", "itemName": "y", "values": yData, "timeStamps": ts }] }).then((data) => {
    console.log(data)
}).catch((err) => {
    console.log(err)
})
```

注意事项：

1. 历史值的类型必须和 item 的 dataType 对应
2. 需要注意的是 GDB 使用的是 unix 的时间戳，所以需要注意不同的语言中时间戳的转换

(3) /page/getRealTimeData

请求描述：获取指定 item 的最近一次更新数据

请求字段：

字段名称	字段描述	字段类型	是否必须
groupNames	分组名称	[]string	是
itemNames	item 名称	[]string	是

请求示例：

```
axios.post(`/data/getRealTimeData`, { "groupNames": ["5DCS", "5DCS"], "itemNames": ["x", "y"] })
{ code: 200, message: '', data: { realTimeData: { x: 1, y: 2 } } }
```

注意事项：

1. 所有获取数据的接口字段的 itemNames 中，itemName 的值不能重复，否则相同的 itemName 获取到的值会被覆盖，即：

```
axios.post(`http://${ip}/data/getRealTimeData`, { "groupNames": ["2DCS", "calc"], "itemNames": ["item1", "item1"]})
```

的结果可能是 {"code":200,"message":"","data":{"realTimeData":{"item1":2}}} 或者 {"code":200,"message":"","data":{"realTimeData":{"item1":1}}}（2DCS 中 item1 的实时值为 1，calc 中 item1 的值为 2）

2.所有的 item 在 gdb 对应的 group 中都必须存在，否则取数就会失败。

3.如果 item 没有实时数据,则返回的实时数据值为 null

例如：点 z 不存在实时值，那么

```
axios.post(`http://${ip}/data/getRealTimeData`, { "groupNames": ["5DCS", "5DCS", "calc"], "itemNames": ["x", "y", "z"] })
{"code":200,"message":"","data":{"realTimeData":{"x":1,"y":2,"z":null}}}
```

(4) /data/ getHistoricalData

请求描述：获取指定 item 的历史数据

请求字段：

字段名称	字段描述	字段类型	是否必须
groupNames	分组名称	[]string	是
itemNames	item 名称	[]string	是
startTimes	开始时间	[]int	是
endTimes	结束时间	[]int	是
intervals	取数间隔	[]int	是

请求示例：

```
'use strict';
const axios = require('axios')
const ip = "192.168.0.199:8082"
const now = new Date(2021, 4, 24, 19, 44, 0)
const st = now.getTime() / 1000 + 8 * 3600
axios.post(`http://${ip}/data/getHistoricalData`, { "groupNames": ["5DCS", "5DCS"], "itemNames": ["x", "y"], "startTimes": [st], "endTimes": [st + 3600], "intervals": [60] }).then(({ data }) => {
    console.log(JSON.stringify(data))
}).catch((err) => {
    console.log(err)
})
```

注意事项：

1. 返回的历史数据格式均为{“itemName”:[[timeStamp],[value]]},第一个数组为时间戳数组，数据类型为 int，第二个数组为对应的值数组，数组中的数据类型与 item 的数据类型一致。
2. 如果在指定的时间内历史不存在，则返回的值为:{“itemName”:[null,null]},即第一个时间戳数组和第二个值数组的值都是 null

   例如：点 z 不存在历史，那么

   ```
   axios.post(`http://${ip}/data/getHistoricalData`, { "groupNames": ["5DCS", "5DCS", "calc"], "itemNames": ["x", "y", "z"], "startTimes": [st], "endTimes": [st + 3600], "intervals": [60] }).then(({ data }) => {
           console.log(JSON.stringify(data))
       }).catch((err) => {
           console.log(err)
   })
   ```

   z 的结果 {"z": [null, null]}
3. 所有的 item 在 gdb 中对应的 group 中必须存在，有一个 item 不存在就会取数失败。

(5) /page/getHistoricalDataWithStamp

请求描述：获取指定 item 在指定时间戳上的历史数据

请求字段：

字段名称	字段描述	字段类型	是否必须
groupNames	分组名称	[]string	是
itemNames	item 名称	[]string	是
timeStamps	unix 时间戳	[][]int	是

请求示例：

```
'use strict';
const axios = require('axios')
```

```
const ip = "192.168.0.199:8082"
const now = new Date(2021, 4, 24, 19, 44, 0)
const st = now.getTime() / 1000 + 8 * 3600
let ts = []
for (var i = 0; i < 60; i++) {
   ts.push(st + i)
}
axios.post(`http://${ip}/data/getHistoricalDataWithStamp`, { "groupNames": ["5DCS", "5DCS"],
"itemNames": ["x", "y"], "timeStamps": [ts,ts]}).then(({ data }) => {
   console.log(JSON.stringify(data))
}).catch((err) => {
   console.log(err)
})
```

注意事项：

1. 返回的数据格式和 getHistoricalData 接口一致。
2. 如果有时间戳对应的历史数据不存在，那么其对应的值数组中的值为 null，需要注意其形式为[null]，而 getHistoricalData 为 null。
3. 所有的 item 在 gdb 中对应的 group 中必须存在，有一个 item 不存在就会取数失败。

(6) /page/getHistoricalDataWithCondition

请求描述：根据指定的条件获取历史数据

请求字段：

字段名称	字段描述	字段类型	是否必须
groupNames	分组名称	[]string	是
itemNames	item 名称	[]string	是
startTimes	开始时间	[]int	是
endTimes	结束时间	[]int	是
intervals	取数间隔	[]int	是
filterCondition	筛选条件	string	是
deadZones	死区条件	[]DeadZone	否

DeadZone 字段：

字段名称	字段描述	字段类型	是否必须
itemName	item 名称	string	是
deadZoneCount	死区值	int	是

请求示例：

```
'use strict';
const axios = require('axios')
const fs = require('fs')
const path = require('path')
const ip = "192.168.0.199:8082"
const now = new Date(2021, 4, 24, 19, 44, 0)
const st = now.getTime() / 1000 + 8 * 3600
axios.post(`http://${ip}/data/getHistoricalDataWithCondition`, { "groupNames": ["5DCS", "5DCS"],
"itemNames": ["x", "y"], "startTimes": [st], "endTimes": [st + 3600], intervals: [10], "filterCondition":
`item["x"] > 2000 && item["y"] > 1000` }).then(({ data }) => {
```

```
    fs.writeFile(path.resolve(__dirname, './fX.txt'), JSON.stringify(data), err => { })
    console.log(JSON.stringify(data))
}).catch((err) => {
    console.log(err)
})
```

注意事项：

1．filterCondition 字段可以是任何合法的 js 表达式，但是对于含有 itemName 的筛选条件，必须写成 item[“itemName”]的形式,如果为”true”则意味着该条件一直成立。

2. deadZones 指定了死区清洗的信息，所谓死区清洗，是指执行的 itemName 中最多允许有多少连续相同的数据。例如{“itemName”: “x”, “deadZoneCount”: 4},即意味着在筛选时，最多允许 x 中有 4 个连续相同的值，即如果 x 有段数据为 0,1,1,1,1,1,2 则返回的数据为 0,1,2。如果 deadZoneCount 的值为 1，则返回的数据为[],<1 或者不传入 deadZones 字段都意味着不使用死区清洗。

3.返回的数据格式即注意事项同 getHistoricalData 接口。