时序数据库使用说明

- 版本：V1.0
- 作者：lazykee
- 发布日期：2024年5月2日
- 更新日期：2024年8月11日
- 发布人：lazykee

开始使用

注意：

> 如果是首次安装，需要首先安装TDengine时序数据库,已经安装过，则无需再次安装
> 数据库路径： C:\ZK\ZkTaosDB-20240524\program\TDengine-server-3.0.7.1-Windows-x64.exe

---

ZK-VIEW 与时序数据库交互方法

注意 需要先运行 `时序数据库` 和 `时序数据库插件`

1. zkview中的数据源地址

```
ws://localhost:9500/tsdb
```

2. zkview与时序数据库交互方法

```
// 请求参数JSON格式， 其他参数参考
const data = {
    "type": "tsdb",  // 类型，必须'tsdb'
    "point": "pointName", // 自定义的中间点名称，为空时，默认不创建中间测点，仅执行方法
    "method": "queryFillData",  // 请求方法名,默认值：queryFillData, 请参考下文-> API方法说明
    "params": {}, // 查询参数，请参考下文-> API方法说明
    "data": [] // 新增数据时，需要提供数据，格式：["dbName,tbName,time,value"]
}
// 全局方法发送请求
window.global.socket.sendData('test', data)
```

- 请求参数说明：

参数名	说明	示例
type	类型，必须'tsdb'	tsdb
point	自定义的中间点名称，为空时，默认不创建中间测点，仅执行方法	pointName
method	请求方法名,默认值：queryFillData	请参考API
params	查询参数	请参考API
data	新增数据时，需要提供数据	请参考添加

ZK-VIEW 与时序数据库交互示例

1. 连接数据源，添加原始点 math1 （自定义的查询点）
2. 添加vue组件

> template

```
<div class="view">
  <el-date-picker
      v-model="value"
      type="datetimerange"
      value-format="yyyy-MM-dd HH:mm:ss"
      placeholder="选择日期时间">
  </el-date-picker>

  <el-button type="primary" @click="search">搜索</el-button>
</div>
```

> script

```
export default {
  data() {
    return {
      value: ['2024-01-02 00:00:00', '2024-01-03 00:00:00']
    }
  },
  mounted() {
    // 必须加一定的延时，否则初始化未完成导致请求失败
    setTimeout(() => {
      this.search()
    }, 100);
  },
  methods: {
    search() {
      const params = {
        "start": this.value[0],
        "end": this.value[1],
        "window": 3600,
        "params": [{
            "dbName": "sim",
            "tbName": "dcs1_10lch30aa101_1",
            "fill": "pre",
            "alias": "point1",
            "fn": "min"
          },
          {
            "dbName": "sim",
            "tbName": "dcs1_10lch30aa101_5",
            "fill": "pre",
            "alias": "point2",
            "fn": "min"
          }
        ]
      }
      const data = {
        "type": "tsdb",
        "point": "math1", //将结果发送到 math1 测点
        "method": "queryFillData",
        "params": params
```

```
      }
      // 全局函数，发送查询请求
      window.global.socket.sendData('test1', data)
    }
  }
}
```

3. 图表展示查询数据，添加图表组件，绑定 math1 测点

> 初始化参数脚本

```
option = {
  backgroundColor: "transparent",
  xAxis: {
    type: 'time'
  },
  yAxis: {
    type: 'value',
    scale: true,
  },
  legend: {},
  dataZoom: [{
    type: 'inside', // 内置型数据区域缩放组件
    zoomOnMouseWheel: true // 是否启用鼠标滚轮进行缩放
  }],
  tooltip: {
    show: true,
    trigger: 'axis', // 'item': 无坐标轴图表中使用，'axis': 有坐标轴使用
    axisPointer: {
      show: true,
      type: 'line', // 'line' 直线指示器,'shadow' 阴影指示器,'none' 无指示器,'cross' 十字准星指示器
      snap: true, // 自动吸附到坐标轴
    }
  },
  series: [{
    type: 'line',
    name: 'point1'
  },
    {
      type: 'line',
      name: 'point2'
    }
  ]
};
```

> 数据更新脚本

```
// bindData["test1.math1"].value        bindData["test1.math1"].time
var data = bindData["test1.math1"].value;
option = {
  series: [{
      data: data.map(item => {
        return [item.time, item.point1]
      })
    },
    {
      data: data.map(item => {
        return [item.time, item.point2]
      })
    }
  ]
};
```

---

4. 表格展示查询数据，添加vue组件，绑定 math1 测点

> template

```
<div class="myTable">
    <vxe-table :data="tableData" max-height="100%" size="small" align="center">
        <vxe-column field="time" title="时间"></vxe-column>
        <vxe-column field="point1" title="point1"></vxe-column>
        <vxe-column field="point2" title="point2"></vxe-column>
    </vxe-table>
</div>
```

> script

```
export default {
  props: [
  'test1_math1', 'test1_math1_time'
  ],
  data() {
    return {
      tableData: []
    }
  },
  watch: {
    test1_math1_time() {
      this.tableData = this.test1_math1
    }
  }
}
```

Python 对接方法示例代码

下载Python示例代码：PyAPI.py

```python
import requests
import json
from datetime import datetime


# 通信方法
class TSDBClient:
    def __init__(self, url="http://localhost:9500/tsdb/"):
        self.url = url

        self.getDb = "getDb"
        self.getTb = "getTb"
        self.getTbCount = "getTbCount"
        self.addData = "addData"  # 新增数字类型数据
        self.addStrData = "addStrData"  # 新增字符串类型数据
        self.queryData = "queryData"
        self.queryTimeRange = "queryTimeRange"
        self.queryLastData = "queryLastData"
        self.queryFirstData = "queryFirstData"
        self.queryTimeData = "queryTimeData"
        self.queryFillData = "queryFillData"

    # 获取全部
    def excutePost(self, method, params):
        try:
            response = requests.post(self.url + method, json=params)
            # 确保响应状态码为200
            if response.status_code == 200:
                return response.json()
            else:
                return {"error": f"HTTP Error with status code {response.status_code}"}
        except Exception as e:
            return {"error": str(e)}


# API测试方法--------------------------------------------------------------------------------
# 测试：新增数据
def addData():
    data = [
```

```
        "dcs1,point1," + current_time + "," + str(100),
        "dcs1,point2," + current_time + "," + str(1000),
        "dcs2,point1," + current_time + "," + str(200),
        "dcs2,point2," + current_time + "," + str(2000),
    ]
    res = tsdb.excutePost(method=tsdb.addData, params=data)
    print(res)


# 测试：获取全部数据库信息
def getDb():
    params = {}
    res = tsdb.excutePost(method=tsdb.getDb, params=params)
    print(res)


# 测试：获取表的记录数
def getTbCount():
    params = {"dbName": "dcs1", "tbName": "point1"}
    res = tsdb.excutePost(method=tsdb.getTbCount, params=params)
    print(res)


# 测试：获取原始数据
def queryData():
    params = {
        "dbName": "dcs1",
        "tbName": "point1",
        "start": "2024-07-01 00:00:00",
        "end": "2024-07-02 00:00:00",
    }
    res = tsdb.excutePost(method=tsdb.queryData, params=params)
    print(res)


# 测试：获取数据库数据时间范围
def queryTimeRange():
    params = {"dbName": "dcs1"}
    res = tsdb.excutePost(method=tsdb.queryTimeRange, params=params)
```

```
    print(res)


# 测试：获取数据库最新数据(等效实时数据)
def queryLastData():
    params = {"dbName": "dcs1"}
    res = tsdb.excutePost(method=tsdb.queryLastData, params=params)
    print(res)


# 测试：获取数据库最早数据(等效初始值)
def queryFirstData():
    params = {"dbName": "dcs1"}
    res = tsdb.excutePost(method=tsdb.queryFirstData, params=params)
    print(res)


# 测试：获取指定测点的某时刻数据
def queryTimeData():
    params = {
        "time": "2024-07-02 00:00:00",
        "params": [
            {"dbName": "dcs1", "tbName": "point1", "alias": "p1"},
            {"dbName": "dcs1", "tbName": "point2"},
        ],
    }
    res = tsdb.excutePost(method=tsdb.queryTimeData, params=params)
    print(res)


# 测试：获取指定测点的指定时间范围内的插值数据
def queryFillData():
    params = {
        "start": "2024-07-01 00:00:00",  # 开始时间
        "end": "2024-07-02 00:00:00",  # 结束时间
        "window": 3600,  # 时间窗口，单位秒
        "params": [
            {
                "dbName": "dcs1",  # 数据库名
```

```
                "tbName": "point1",  # 表名
                "fill": "pre",  # 填充方式，可选值：pre, linear
                "alias": "p1",  # 别名 - 非必填
                "fn": "min",  # 聚合函数，可选值：min, max, avg, sum, first, last
            },
            {
                "dbName": "dcs1",
                "tbName": "point2",
                "fill": "pre",
                "alias": "p2",
                "fn": "min",
            },
        ],
    }
    res = tsdb.excutePost(method=tsdb.queryFillData, params=params)
    print(res)


# 测试：新增字符串类型数据
def addStrData():
    # 连接数据源
    tsdb = TSDBClient()
    json_data = {
        "name": "Alice",
        "age": 30,
        "city": "上海",
        "has_children": False,
        "children": None,
    }
    data = [
        {
            "dbName": "strdb",
            "tbName": "str_point1",
            "time": "2024-08-10 16:03:00",
            "value": json.dumps(json_data, ensure_ascii=False),
        },
        {
            "dbName": "str",
            "tbName": "str_point1",
```

```
            "time": "2024-08-10 16:04:00",
            "value": json.dumps(json_data, ensure_ascii=False),
        },
    ]
    res = tsdb.excutePost(method=tsdb.addStrData, params=data)
    print(res)


# 算法对接方式
# 1. 连接数据源
# 2. 获取数据
# 3. 计算数据
# 4. 将计算结果通过addData方法写入数据库
# （计算结果如果是数字类型数据，则通过addData方法写入数据库，如果是字符串类型或者对象类型数据，则通过addSt
# addData方法与addStrData方法不可以在同一个数据库中混用，addStrData需要指定新的数据库名称，专门用于存储字

# 主程序
if __name__ == "__main__":
    # 示例代码
    # 1. 连接数据源
    tsdb = TSDBClient()
    # 2. 获取数据
    params = {
        "start": "2024-07-01 00:00:00",  # 开始时间
        "end": "2024-07-02 00:00:00",  # 结束时间
        "window": 3600,  # 时间窗口，单位秒
        "params": [
            {
                "dbName": "dcs1",  # 数据库名
                "tbName": "point1",  # 表名
                "fill": "pre",  # 填充方式，可选值：pre, linear
                "alias": "p1",  # 别名 - 非必填
                "fn": "min",  # 聚合函数，可选值：min, max, avg, sum, first, last
            },
            {
                "dbName": "dcs1",
                "tbName": "point2",
                "fill": "pre",
                "alias": "p2",
```

```
            "fn": "min",
        },
    ],
}
res = tsdb.excutePost(method=tsdb.queryFillData, params=params)
print(res)

current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
# 3. 计算数据
# 假设一系列算法后得到计算结果是单个的数值类型，将计算结果通过addData方法写入数据库
predict_value = 1000
data = ["dcs1,predict_value," + current_time + "," + str(predict_value)]
tsdb.excutePost(method=tsdb.addData, params=data)

# 假设一系列算法后得到计算结果是对象类型，包含多个参数，将计算结果通过addStrData方法写入数据库
predict_json_data = {"name": "Alice", "age": 30, "city": "上海"}
data = [
    {
        "dbName": "strdb",  # 数据库名
        "tbName": "str_point1",  # 表名
        "time": "2024-08-10 16:03:00",  # 时间
        "value": json.dumps(
            predict_json_data, ensure_ascii=False
        ),  # 字符串类型数据
    }
]
tsdb.excutePost(method=tsdb.addStrData, params=data)
```

API方法说明

1.获取指定测点的指定时间范围内的插值历史数据

- 方法名称: queryFillData
- **该方法仅针对数值类型的数据查询有效，不支持字符串查询**
- 请求参数示例:

```
{
  "start": "2024-01-01 00:00:00",
  "end": "2024-01-01 00:00:00",
  "window": 1,
  "params": [
    { "dbName": "simulate", "tbName": "point1", "fill": "pre", "alias": "myPoint1", "fn": "avg"},
    { "dbName": "simulate", "tbName": "point2", "fill": "liner", "alias": "myPoint2", "fn": "last"
  ]
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
start	开始时间	2024-01-01 00:00:00	是	String
end	结束时间	2024-01-01 00:00:00	是	String
window	聚合窗口时间间隔(秒)	1	否	Int
params	请求参数列表	[{"dbName": "simulate", "tbName": "point1", "alias": "myPoint", "fn": "avg"}]	是	List

- parms参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String
tbName	数据表(点位)名称	point1	是	String
fill	填充类型 前向填充(pre) 线性(liner)	pre	否	String
alias	返回别名 默认值为 tbName	myPoint	否	String
fn	聚合函数,默认值(avg)	avg	否	String

参数名	说明	示例值	必须	类型
	window不为空时生效 (min)窗口最小值 (max)窗口最大值 (avg)窗口平均值 (sum)窗口总和 (first)窗口第一个值 (last)窗口最后一个值			

- 返回值示例:

```
[
    {"point1": 0.0, "point2": 0.0, "time": "2024-01-01 00:00:00"},
    {"point1": 1.0, "point2": 1.0, "time": "2024-01-01 00:00:01"}
]
```

- 返回值说明:

参数名	说明	类型
time	时间	String
xxx	表(点位)名称	Double

2.获取数据库信息

- 方法名称: getDb
- 请求参数: 无
- 返回值示例:

```
[
    {"dbName": "sim", "duration": 365},
    {"dbName": "zk1", "duration": 3650}
]
```

- 返回值说明:

参数名	说明	类型
dbName	数据库名称	String
duration	数据时间范围(秒)	Int

3.获取表信息

- 方法名称: getTb
- 请求参数示例:

```
{
  "dbName": "simulate"
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String

- 返回值示例:

```
[
  "dcs1_001_00765_out",
  "dcs1_007_04200_out",
  "dcs1_011_00338_out",
  "dcs1_011_06756_out"
]
```

- 返回值说明:

参数名	说明	类型
xxx	表(点位)名称	String

4.获取表记录数

- 方法名称: getTbCount
- 请求参数示例:

```
{
  "dbName": "simulate",
  "tbName": "point1"
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String
tbName	数据表(点位)名称	point1	是	String

- 返回值示例:

```
1000
```

- 返回值说明:

参数名	说明	类型
(空)	记录数	Int

5.添加数据(数值类型数据)

- 方法名称: addData
- 请求参数示例:

```
[
  "simulate,point1,2024-01-01 00:00:00, 0",
  "simulate,point1,2024-01-01 00:00:01, 1",
  "simulate,point2,2024-01-01 00:00:00, 10",
  "simulate,point2,2024-01-01 00:00:01, 15"
]
```

- 参数说明:

参数名	说明	示例值	必须	类型
(空)	数据列表	["dbName,tbName,time,value"]	是	List

- 数据列表说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String
tbName	数据表(点位)名称	point1	是	String
time	时间	2024-01-01 00:00:00	是	String
value	值	0	是	Double

- 返回值： 无

6.添加数据(字符串类型数据)

- 方法名称: addStrData
- **该方法仅针对数值类型的数据查询有效，不支持字符串查询**
- **addData方法与addStrData方法不可以在同一个数据库中混用，addStrData需要指定新的数据库名称，专门用于存储字符串类型数据**
- 请求参数示例:

```
[
  {
    "dbName": "str",
    "tbName": "a",
    "time": "2024-08-10 16:03:00",
    "value": "Json字符串1"
  },
  {
    "dbName": "str",
    "tbName": "a",
    "time": "2024-08-10 16:04:00",
    "value": "Json字符串2"
  }
]
```

- 参数说明:

参数名	说明	示例值	必须	类型
(空)	数据列表	[{}, {}]	是	List

- 数据列表说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String
tbName	数据表(点位)名称	point1	是	String
time	时间	2024-01-01 00:00:00	是	String
value	值	0	是	Double

- 返回值： 无

7.获取原始数据

- 方法名称: queryData
- 请求参数示例:

```
{
  "dbName": "simulate",
  "tbName": "point1",
  "start": "2024-01-01 00:00:00",
  "end": "2024-01-01 00:00:00"
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String
tbName	数据表(点位)名称	point1	是	String
start	开始时间	2024-01-01 00:00:00	是	String
end	结束时间	2024-01-01 00:00:00	是	String

- 返回值示例:

```
[
  {"time": "2024-01-01 00:00:00", "value": 0.0},
  {"time": "2024-01-01 00:00:01", "value": 1.0}
]
```

- 返回值说明:

参数名	说明	类型
time	时间	String
value	值	Double

8.获取数据库数据时间范围

- 方法名称: queryTimeRange
- 请求参数示例:

```
{
  "dbName": "simulate"
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String

- 返回值示例:

```
["2024-07-04 15:54:07","2024-07-04 15:58:52"]
```

- 返回值说明:

参数名	说明	类型
索引0	开始时间	String
索引1	结束时间	String

9.获取数据库最新数据(等效实时数据)

- 方法名称: queryLastData
- 请求参数示例:

```
{
  "dbName": "simulate"
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String

- 返回值示例:

```
[
    { "name": "point1", "time": "2024-07-04 16:00:54", "value": 35.0},
    { "name": "point2", "time": "2024-07-04 16:00:54", "value": 27.831657459280922}
]
```

- 返回值说明:

参数名	说明	类型
name	表(点位)名称	String
time	时间	String
value	值	Double

10.获取数据库最早数据(等效初始值)

- 方法名称: queryFirstData
- 请求参数示例:

```
{
  "dbName": "simulate"
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String

- 返回值示例:

```
[
    { "name": "point1", "time": "2024-07-04 16:00:54", "value": 35.0},
    { "name": "point2", "time": "2024-07-04 16:00:54", "value": 27.831657459280922}
]
```

- 返回值说明:

参数名	说明	类型
name	表(点位)名称	String
time	时间	String
value	值	Double

11.获取指定测点的某时刻数据

- 方法名称: queryTimeData
- 请求参数示例:

```
{
  "time": "2024-07-02 00:00:00",
  "params": [
    {"dbName": "simulate", "tbName": "point1", "alias": "myPoint"}
  ]
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
time	指定时间	2024-01-01 00:00:00	是	String
params	请求参数列表	[{"dbName": "simulate", "tbName": "point1", "alias": "myPoint"}]	是	List

- params参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	simulate	是	String
tbName	数据表(点位)名称	point1	是	String
alias	返回别名	myPoint	否	String

- 返回值示例:

```
{
  "myPoint2":27.0,
  "myPoint1":1.0
}
```

- 返回值说明:

参数名	说明	类型
xxx	指定时间测点xxx的值	Double

12.获取指定测点的统计数据

- 方法名称: queryCountData
- 请求参数示例:

```
{
  "start": "2024-08-09 11:26:06",
  "end": "2024-08-09 12:13:55",
  "params": [
    {"dbName": "redis", "tbName": "point1",  "fn": "avg"},
    {"dbName": "redis", "tbName": "point2", "alias": "countP2", "fn": "count"}
  ]
}
```

- 参数说明:

参数名	说明	示例值	必须	类型
start	开始时间	2024-01-01 00:00:00	是	String
end	结束时间	2024-01-01 00:00:00	是	String
params	请求参数列表	[{"dbName": "redis", "tbName": "point1", "fn": "avg"}]	是	List

- params参数说明:

参数名	说明	示例值	必须	类型
dbName	数据库名称	redis	是	String
tbName	数据表(点位)名称	point1	是	String
alias	返回别名	myPoint	否	String
fn	聚合函数,默认值(avg) (AVG)统计指定字段的平均值 (COUNT)统计指定字段的记录行数 (SPREAD)统计表中某列的最大值和最小值之差 (STDDEV)统计表中某列的均方差 (SUM)统计表中某列的和 (HYPERLOGLOG)采用 hyperloglog 算法，返回某列的基数	avg	否	String

- 返回值示例:

```
{
  "point1":1353.6666666666667,
  "point2":431.6451732100746
}
```

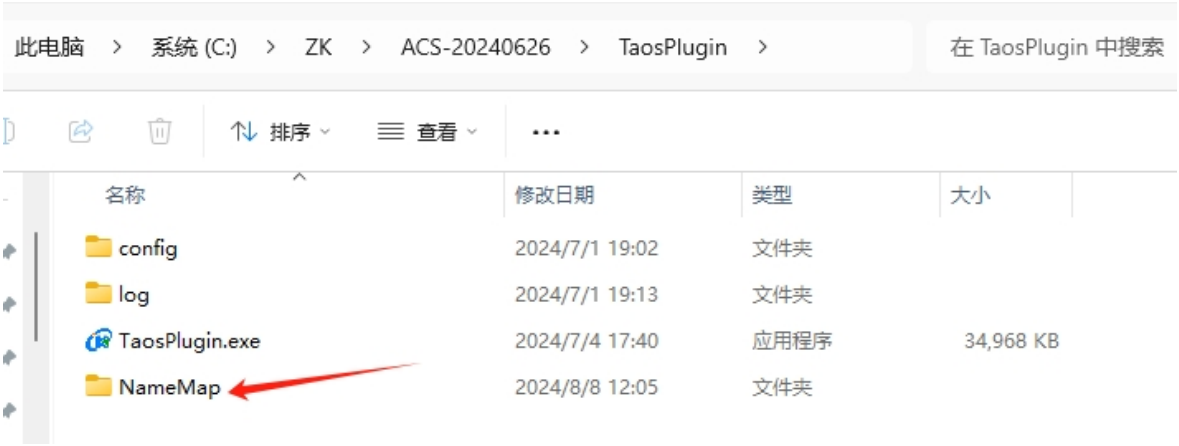
- 返回值说明:

参数名	说明	类型
测点名称（如果指定alias别名，则为别名，否则为表名）	统计结果	Double

补充

为测点添加中文描述

1. 打开插件运行的根目录，进入 `NameMap` 文件夹



2. 根据已提供模板 `Demo.csv` 文件模板，填写中文名称和对应的数据库表名称(编码格式为UTF-8编码)
   下载Demo.csv

`不建议用WPS保存，可能会出现乱码问题`

支持任意名称的多个NameMap配置文件，系统会自动读取NameMap下所有的文件

3. 在文件中按照如下格式写入对应的测点名称和测点描述

如果是标准的文档，如随机数测点文档，则直接将配置文件到该文件夹即可，系统自动读取前三列（描述、库名、表名）

描述	数据库名称(必填)	点名(必填唯一)
测点1	simulate	point1
测点2	simulate	point2
测点3	simulate	point3
测点4	simulate	point4

描述	数据库名称(必填)	点名(必填唯一)
测点5	simulate	point5

第一列为显示的中文名称，第二列为数据库中库的名称，第三列为数据库中表的名称，需要严格对应

4. 打开zkview数据源管理>原始点管理, 选择对应的数据库，重新进行测点保存