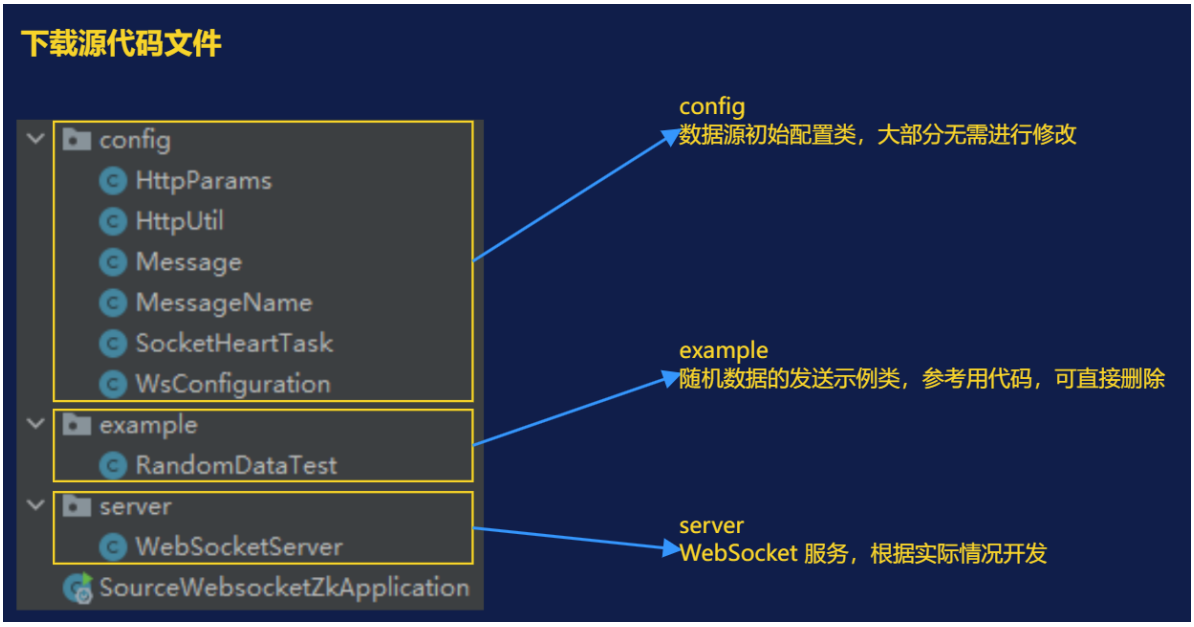
数据源使用说明书

zkview全局通过WebSocket协议实时推送数据，无需轮询，实时性高。
提供了Java、Python与zkview的通信源代码，通过对通信源码的二次开发，即可实现数据源接入。

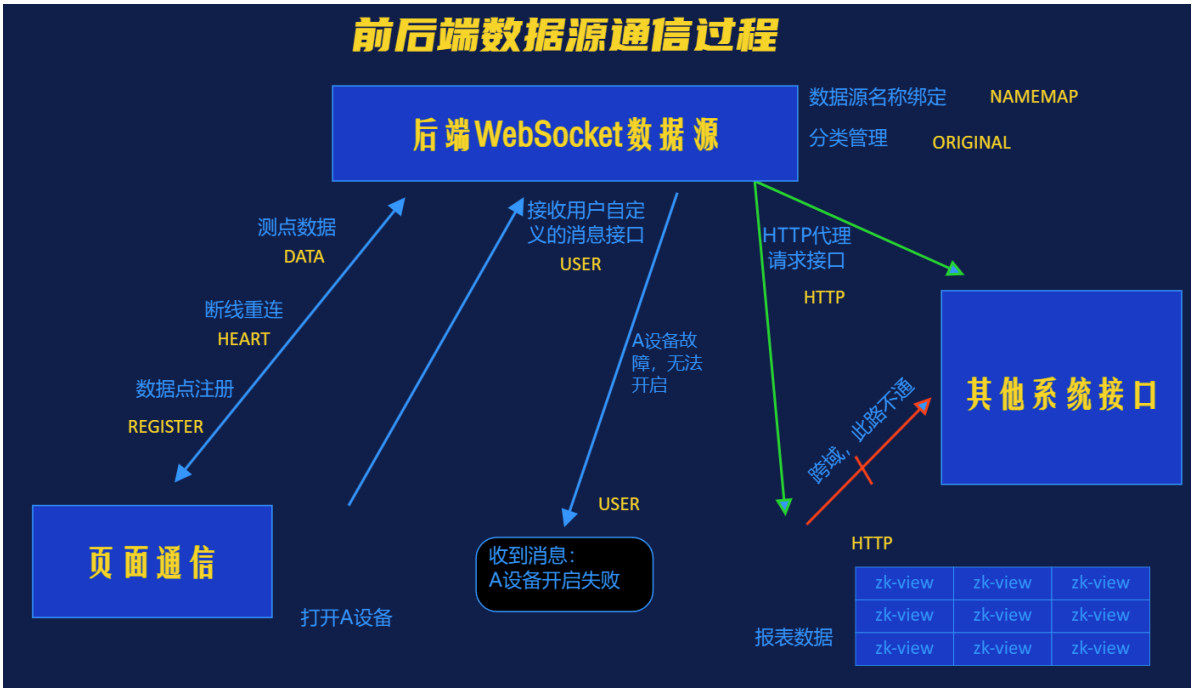
1. 数据源下载

登录官网：http://zkview.com，使用手机号码注册，切换到下载栏，点击Java数据源下载。

2. 数据源源码介绍



3. 通信过程介绍



> zkview页面与数据源通过WebSocket协议建立连接，连接成功后，zkview页面会接收到数据源推送的实时数据。
> 通信过程通过关键字进行定义，关键字包括：HEART、REGISTER、DATA、NAMEMAP、ORIGINAL、USER、HTTP。
>
> > HEART :【前后端识别】发送心跳的 name 标志 , 用于断线重连
> > DATA :【前端识别】发送测点数据的 name 标志 , 测点列表自动解析并显示
> > ORIGINAL :【前后端识别】数据源管理全部测点数据获取 分区数据， 用于数据源/原始点管理，进行测点添加中，分许下拉框选择
> > ORIGINALDATA :【前端识别】数据源管理全部测点数据获取的所有测点信息， 用于数据源/原始点管理，进行测点选择添加
> > NAMEMAP :【前端识别】数据源管理全部测点 key 对应的中文名称 name 标志， 用于数据源/原始点管理，测点中文名称显示添加
> > REGISTER :【后端识别】注册节点消息 name 标志， 前端在图纸预览或部署打开时，自动发送图纸中添加的原始点，后端可对测点进行筛选后发送，以达到节约计算机资源目的
> > HTTP :【前后端识别】HTTP 代理请求 name 标志，前后端通过该标志进行匹配识别
> > USER :【前后端识别】用户自定义数据 name 标志，自定义数据可在数据源列表中的事件简本进行捕获
> > STATUS :【前端识别】用户自定义数据 设备离线 标志，通过该字段发送离线设备测点信息，前端收到后会进行对应处理
>
> 
>

通信源码如下：

```
switch (msgName) {
    /** 数据源心跳监听接口
     该接口用户切勿修改，是数据源心跳的监听接口
     */
    case MessageName.HEART -> {
        sendMessage(this, new Message(MessageName.HEART, null), false);
        this.heartTime = System.currentTimeMillis();
    }


    /** 全部测点数据获取接口（用户数据源管理页面，原始测点弹出面板，进行测点勾选添加）
     *       请在该接口中返回全部的测点信息
     *       注意：  该接口在原始点管理打开后，会收到数据，通知浏览器所有测点信息，进行测点添加
     */
```

```
    case MessageName.ORIGINAL -> {
        // 示例：可以配置数据源分类列表，并在网页原始点管理中进行显示
//                List<Message> dataList = new ArrayList<>();
//                dataList.add(new Message("1号区域", "area1"));
//                dataList.add(new Message("2号区域", "area2"));
//                dataList.add(new Message("3号区域", "area3"));
//                sendMessage(this, new Message(MessageName.ORIGINAL, dataList),
false);
//
//                // 示例：获取设置的区域参数并，发送测点数据到当前用户
//                String area = (String) msgData;
//                if ("area1".equals(area)) {
//                    // 发送数据对应的测点名称
//                    sendMessage(this, new Message(MessageName.NAMEMAP,
RandomDataTest.nameMap), false);
//                    // 发送测点的实时值
//                    sendMessage(this, new Message(MessageName.ORIGINALDATA,
RandomDataTest.data), false);
//                }

        // 不分项目类别的情况下，直接发送全部测点示例
        sendMessage(this, new Message(MessageName.NAMEMAP,
RandomDataTest.nameMap), false);
        sendMessage(this, new Message(MessageName.ORIGINALDATA,
RandomDataTest.data), false);
    }


    /** 注册测点登记接口
     *      该接口用户切勿修改，可进行逻辑处理
     *      接收到的值为：图纸上对该数据源的所有注册点位信息
     *                    用户可根据注册的点位信息筛选发送的数据
     *      注意： 每一个数组对应一个测点，拼接起来即为完整测点key，如收到注册点["c","b"]
["c","cc","ccb"]，即为注册两个测点，分别为 c.b 和 c.cc.ccb
     *      可以根据每个用户注册的点，单独发送注册点数据，以节约服务器资源
     */
    case MessageName.REGISTER -> {
        System.out.println("收到用户注册测点");
        List<List<String>> registerPointList = (List<List<String>>) msgData;
        System.out.println(JSON.toJSONString(msgData));
        // 示例添加一级注册点，仅发送一级注册点数据
        for (List<String> point : registerPointList) {
            registerSet.add(point.get(0));
        }
        // 收到注册消息，立刻回复所有的测点信息，这时页面刷新时，可以快速加载所有的数据
        sendMessage(this, new Message(RandomDataTest.data), true);
    }



    /** HTTP 代理数据源
     *  该接口封装了代理的 HTTP 请求接口
     *      封装目的：没有同源限制，客户端可以与任意服务器通信
     *      返回的数据用户可通过 sendMessage() 方法向页面输出
     */
```

```
    case MessageName.HTTP -> {
        System.out.println("收到 HTTP 请求");
        HttpParams httpParams = JSON.parseObject(JSON.toJSONString(msgData),
HttpParams.class);
        String httpReturnData = HttpUtil.http(httpParams);
        System.out.println(httpReturnData);

        // 示例: 将定义key的http请求将返回值，发送回页面返回
        String key = httpParams.getKey();
        if (key != null && !"".equals(key)) {
            Map<String, Object> map = new HashMap<>();
            map.put("key", key);
            map.put("data", httpReturnData);
            Message httpMessage = new Message(MessageName.HTTP, map);
            sendMessage(this, httpMessage, false);
        }
    }


    /** 用户发送数据
     * 接收页面写入的websocket数据，用户可自行数据进行后续处理
     */
    case MessageName.USER -> {
        System.out.println("收到用户发送数据");
        // 自定义处理逻辑（如身份认证等需求）
        System.out.println(msgData);
        sendMessage(this, new Message(MessageName.USER, "收到了网页发送的数据（" +
msgData + ")"), false);
    }

    /**
     *  未匹配消息时，进行打印
     */
    default -> {
        System.err.println("未匹配类型数据");
    }

}
```

4. 数据推送

通过调用sendMessage()方法，向页面推送数据，数据格式为Message对象，包含消息名称和消息内容。

`WebSocketServer.sendMessage(new Message(data), false);`

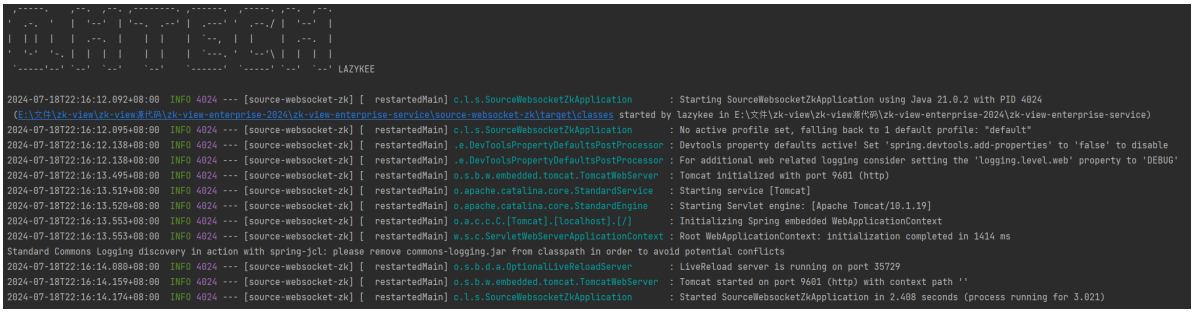
> new Message(data) : 发送的数据内容，格式为Map<String, Object>，key为测点名称，value为测点实时值。
> Message默认类型为DATA类型。
> 第二个参数为true时，表示推送给用户的数据为用户在zkview原始点管理选中的数据，false表示推送给用户全部的数据。
> 项目内置了RandomDataTest类，用于生成随机数据，用户可根据需求修改。

```
@Component
```

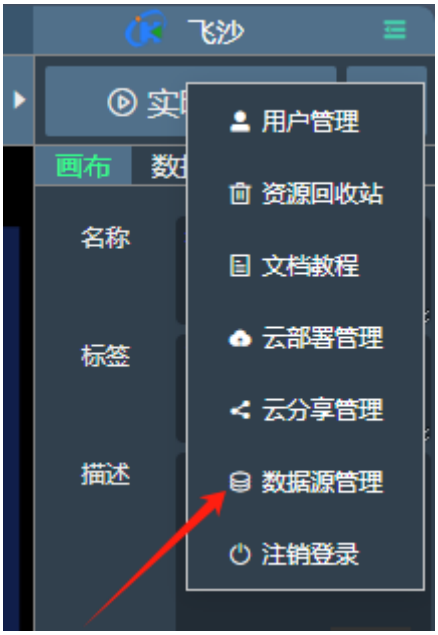
```java
public class RandomDataTest{

    public static Map<String, Object> data = new HashMap<>();

    public static Map<String, String> nameMap = new HashMap<>();

    // 测点对应的中文名称，可进行原始点添加时，自动显示
    static {
        nameMap.put("point1", "0-100随机值浮点数");
        nameMap.put("point2", "0-100随机值整数");
        nameMap.put("case1", "子节点1对象数据");
        nameMap.put("case1.name", "子节点1字符串名称");
        nameMap.put("case1.value", "子节点1数据0-10浮点数");
    }
    /**
         {
             "point1": 42.9699627696726,
             "point2": 22,
             "case1": {
                 "name": "实时值7.608865965560662",
                 "value": 1.1397281906438461
             }
         }
     */
    @Scheduled(fixedRate = 1000)
    public void sendRandomData() throws IOException {
        data.put("point1", Math.random() * 100);
        data.put("point2", Math.round(Math.random() * 100));
        Map<String, Object> case1 = new HashMap<>();
        data.put("case1", case1);
        case1.put("name", "实时值" + Math.random() * 10);
        case1.put("value", Math.random() * 10);
        // 发送示例数据
        WebSocketServer.sendMessage(new Message(data), false);
    }
}
```

5 内置随机数据源使用方法

1. 启动数据源项目



2. 打开zkview，切换到数据源管理页面

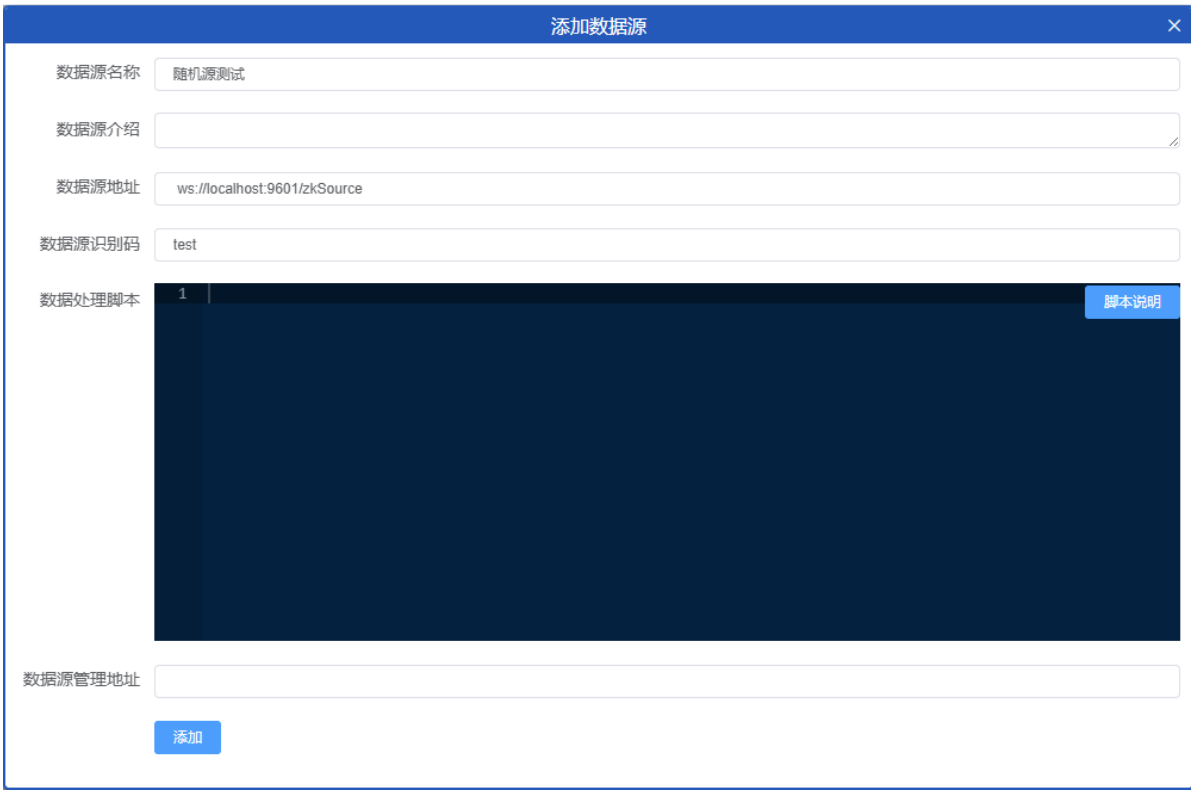


3. 添加项目

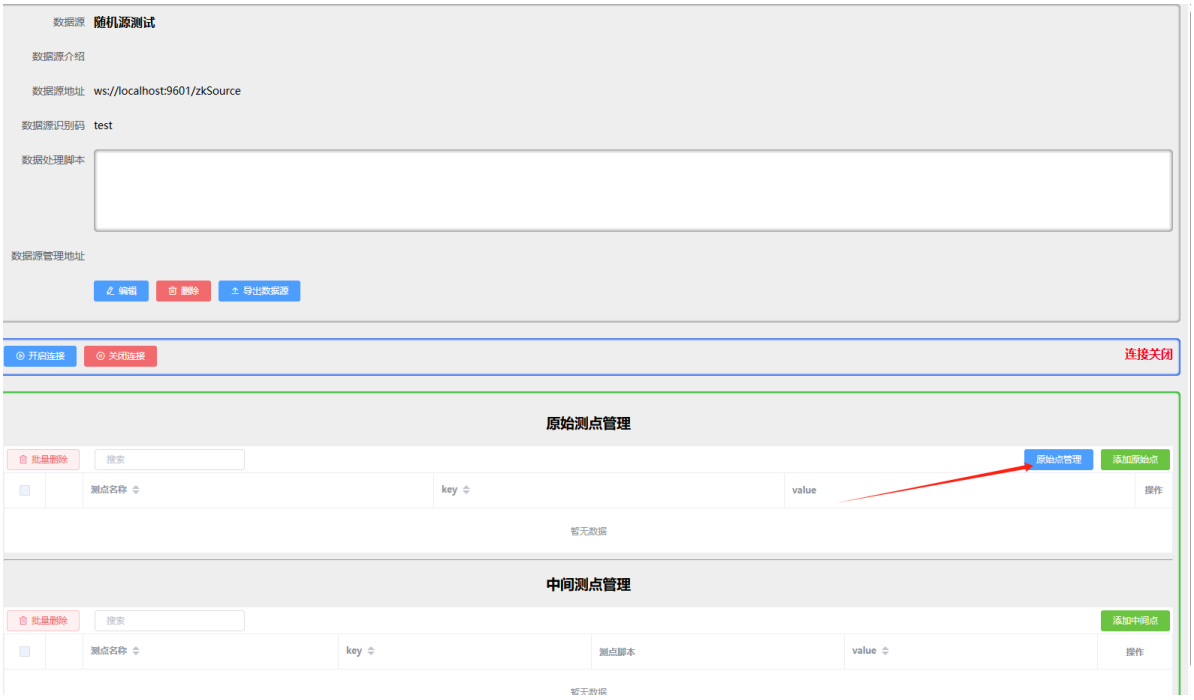
4. 添加数据源

5. 填写数据源信息

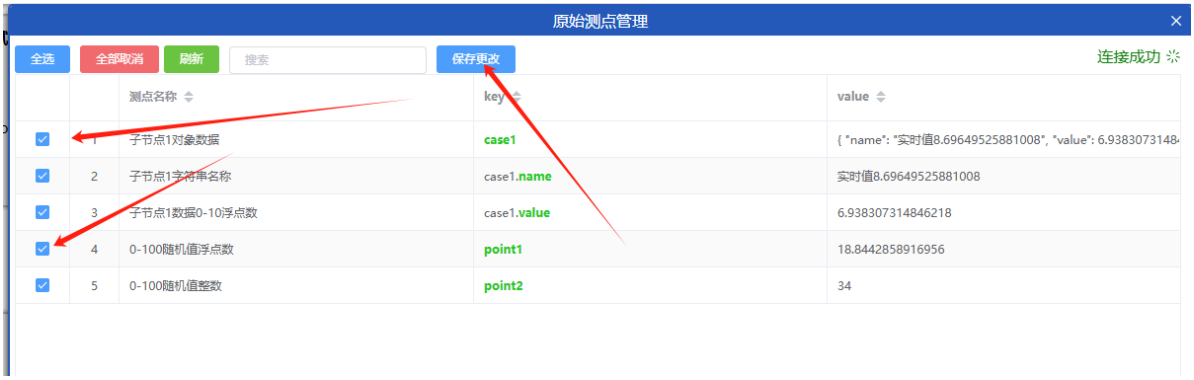
> 数据源地址默认为：`ws://localhost:9601/zkSource`，数据源识别码必填，点击添加



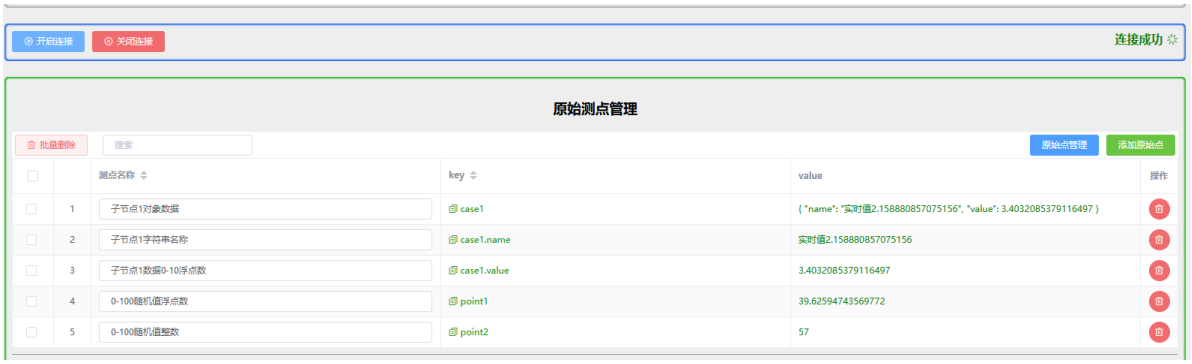
6. 点击添加的数据源，点击原始点管理，添加原始点



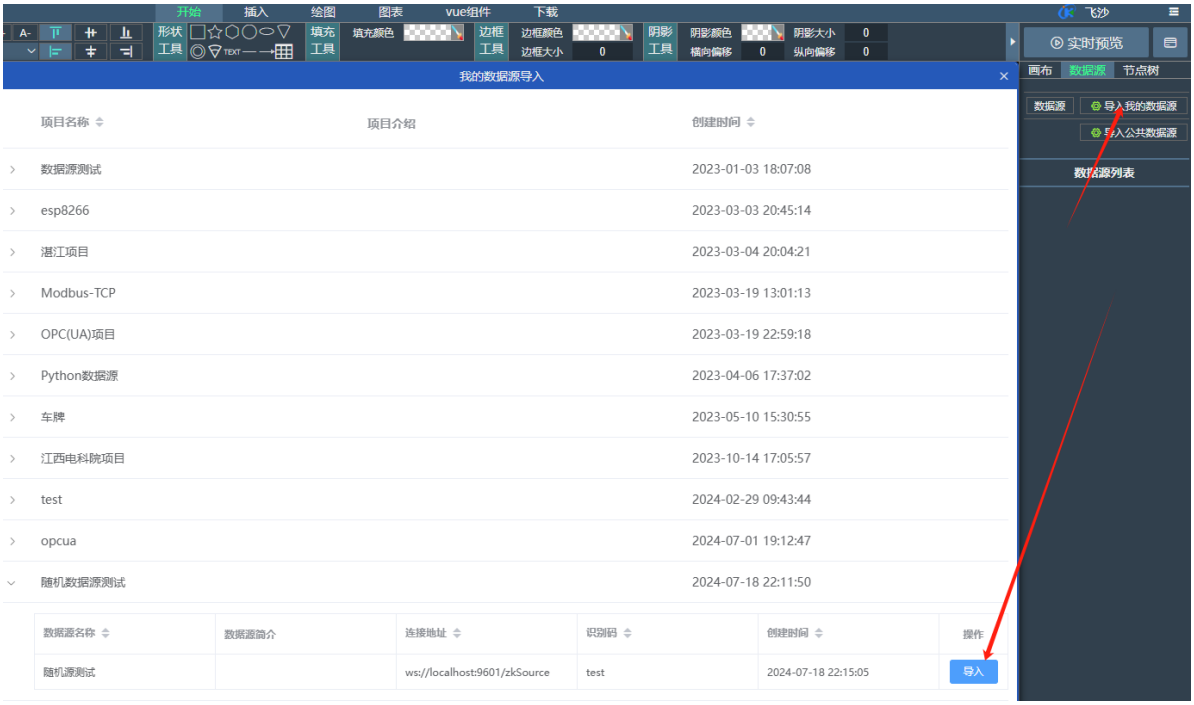
7. 选择数据源自动推送过来的测点，选择需要的测点进行勾选，点击保存更改



8. 点击开启连接，查看数据变化情况，正常情况，显示连接成功，并且数据发生变化



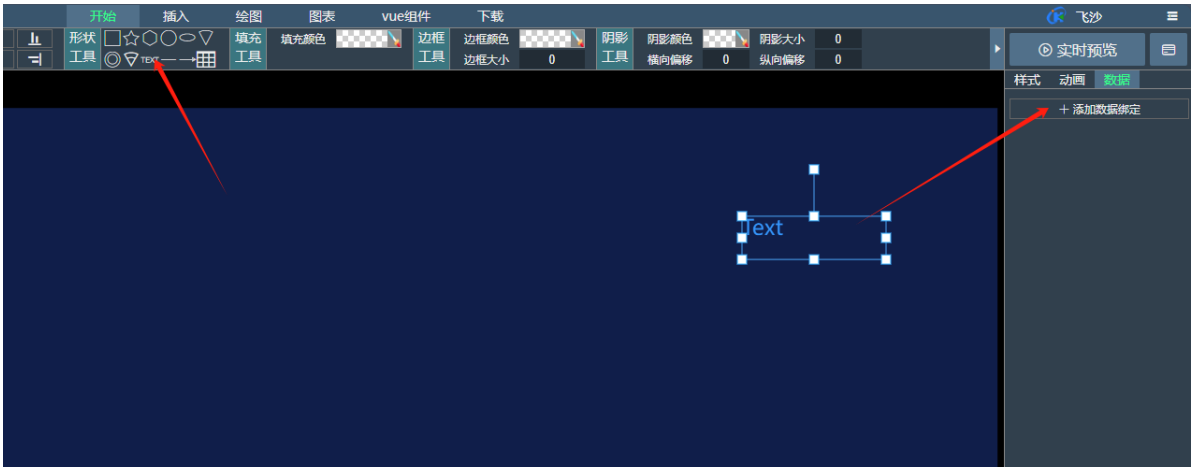
9. 关闭连接，切换到图纸页面，点击图纸空白处，切换到数据源面板，点击导入我的数据源，选择刚才添加的数据源，点击导入



10. 点击导入后，数据源显示导入连接信息

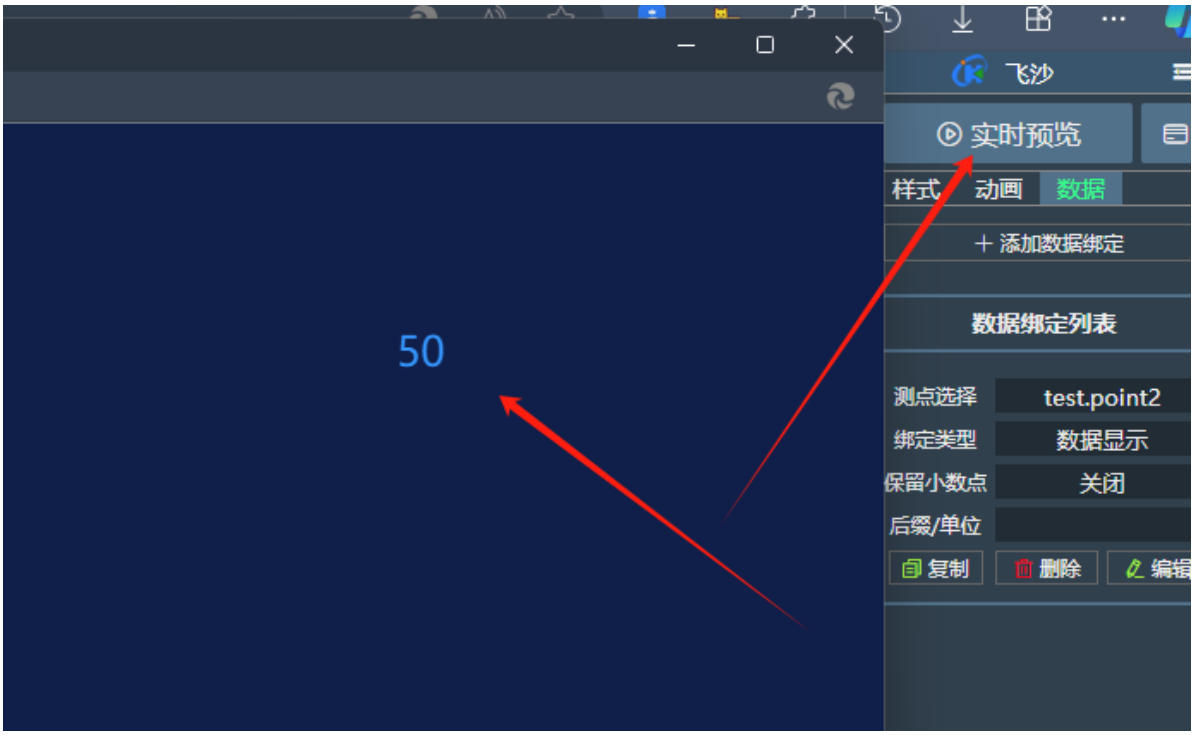
11. 添加数据绑定，显示实时数据

> 拖拽文本元素到图纸中，切换到数据面板，点击 添加数据绑定



12. 点击选择测点，选择需要的测点，最后点击添加按钮

13. 点击实时预览，观察数据变化



6 对接实际的数据

模拟随机数据源的代码结构，删除随机数的逻辑，替换为实际的数据读取方法。读取数据后：

> 1. 将数据结构整理为 Map<String, Object>，key为测点名称，value为测点实时值。
> 2. 将新的数据存入 Map<String, Object> data，当图纸页面打开或刷新时，会自动调用缓存的数据
> 3. 调用WebSocketServer.sendMessage(new Message(data), false); 向页面推送数据。

```
// 缓存实时数据，当图纸加载时，自动调用data中的数据，将实时值显示到图纸中
public static Map<String, Object> data = new HashMap<>();
// 测点对应的中文名称，可进行原始点添加时，自动显示，如果不需要显示中文名称，可不用填写该变量。
public static Map<String, String> nameMap = new HashMap<>();
```

7 MQTT例程

使用过程中，可以参考官网提供的MQTT数据源通信源码，同样基于此结构进行开发，可供开发参考。

> 项目中提供Python与zkview源通信源码，如果使用Python开发，可以参考该源码进行通信程序开发。