版本编号	*变化状态	变更内容和范围	变更日期	变更人	批准日期	批准人
1.0.0	A	新增	2022-7-8	高玉坤		程揭章

> *变化状态：A——增加，M——修改，D——删除，N——正式发布

1 引言

1.1 执行摘要

- 为智慧时空大数据平台建立统一的数据订阅服务，不同的客户端均使用统一的方式对数据进行订阅。作为第三方应用程序开发的依据和输入。

2 公共说明

2.1 接入地址

接口分类	接入地址
数据订阅	ws://geovis.center.com/wsserver

2.1 签名认证

说明

- 调用方需要对发送到ws的请求进行签名，执行签名计算的签名值需要赋值到URL的sign属性，以便服务端进行签名验证。

参数介绍

- **待签名字符串为**： accesskeyId + accesskeySecret + timestamp；
- **accesskeyId**：授权账户，40位以内字符,智慧创新研究院智慧中台全局唯一；
- **accesskeySecret**：在智慧创新研究院智慧中台申请的accesskeySecret，不能明文发送；
- **timestamp**：Unix时间戳，精确到毫秒；

算法

- 签名算法就是对待签名字符串计算32位小写SHA-256值，算法示例见附录。

3 数据订阅服务接口

3.1 数据通信模型

- 客户端向服务端发起连接请求，服务端对连接请求参数进行鉴权，鉴权失败，则断开连接并返回客户端鉴权失败信息；鉴权成功，则建立通道。
- 鉴权成功后，客户端可多次向服务端发起订阅关系，多次订阅同一个topic的key组合，服务端则自动合并为一个订阅，默认为同一个客户端订阅；

- 针对同一个云服务，发起多次连接，服务端则认为是多个客户端发起请求，所有消息将平衡分发到多个客户端上，每个云服务最多可发起连接数为系统可配置参数。
- 取消订阅，则删除订阅关系，服务端返回客户端取消响应结果。

3.2 建立连接

- 提供建立连接的功能。客户端发起连接请求，数据订阅系统会对客户端连接请求进行鉴权后，建立与客户端的连接。

3.2.1 建立连接

功能描述

- 客户端发起连接请求，请求时携带对应应用的accesskeyId和accesskeySecret签名信息进行鉴权，鉴权成功返回鉴权结果，鉴权失败，则返回错误信息并断开连接。

接入地址：/websocket

输入参数

参数名	类型	位置	必填	说明
accessKeyId	String	URL	必填	授权ID，平台全局唯一。
sign	String	URL	必填	对请求进行签名运算产生的签名,签名算法见2.5章节。
timestamp	String	URL	必填	Unix时间戳，精确到毫秒。
resetTime	Integer	URL	选填	指定消息消费的时间，以分钟为单位。取值范围：可以最多指定到2个小时之前的数据（0-120）；不传当前参数时，从最后一次消费的时间开始消费; 当传入时间时，从指定时间开始消费消息。如：传入0时，从当前时间开始消费；传入60，从当前时间1小时前开始消费。

输出参数

参数名	类型	位置	必填	说明
cmd	String	Body	必填	操作说明
data	String	Body	必填	响应结果

请求样例

接口名称：消息订阅接口

请求地址：ws://geovis.center.com/wsserver/websocket?accesskeyId=3o80mxreadyagomr&timestamp=1491013448629&sign=c70500c16563b5ccc7d032831bff34a5cb02c147ca6beeffff54d22d262a319e
用户请求：无
请求应答：

```
{
"cmd": "authenticate-ack",
"data": {
"code": "00000",
"result": "success"
}
}
```

状态码：

```
00000 ：成功
00001 ：失败
```

3.2.2 订阅接口

功能描述

- 订阅指定topic消息，消息订阅必须在建立连接成功的前提下进行，如果建立连接返回成功，才可以发送订阅，如果失败，则无法进行订阅。

接口定义

接口名称：订阅接口

客户端向云端发送的JSON字符串格式数据如下（红色部分为示例数据）：

```
{
"cmd": "subscribe",
"topics": ["*"]
}
```

说明：

- (1) 客户端一个连接情况下只能发起一次订阅消息(服务器端做了限制，多发不起作用)，订阅信息中的多个topic，其中如果有任何一个订阅验证失败，则本次请求全部订阅均失败，只有当全部topic的keys订阅成功，则本次订阅成功。
- (2) *代表统配所有topic的意思

云端向客户端返回订阅结果的响应JSON字符串格式数据如下（红色部分为示例数据）：

```
{
"cmd": "subscribe-ack",
"data": {
```

```
        "code": "00000",
        "result": "success",
        "desc": "subscribed ok"
        }
        }
```

订阅成功之后，当topic有数据更新时，客户端会收到如下JSON字符串格式数据：

```
        {
        "partition": "1",
        "data": "
{\"msg\":\"1111111111\",\"gid\":\"1111111111\",\"t\":\"1111111111\",\"name\":\"土壤水
TDS\",\"id\":\"1111111111\",\"gname\":\"田间气象端口1\",\"value\":\"327\"}",
        "topic": "SocketData01",
        "time": "2022-07-12 13:33:07"
        }
```

错误码：

- 34001,34002,34003,34004,34005,34006,34999

3.2.3 取消订阅接口

功能描述

- 取消订阅指定topic消息，取消订阅必须在建立连接成功的前提下进行，如果建立连接返回成功，才可以发取消订阅，如果失败，则无法进行取消订阅。

接口定义

接口名称：取消订阅接口

客户端向云端发送的JSON字符串格式数据如下（红色部分为示例数据）：

```
        {
        "cmd": "unsubscribe",
        "topics": ["DEV_STATUS",
        "DEV_BIGDATA"]
        }
```

云端向客户端返回订阅结果的响应JSON字符串格式数据如下（红色部分为示例数据）：

```
        {
        "cmd": "unsubscribe-ack",
        "data": {
        "code": "00000",
        "result": "success",
        "desc": "unsubscribed ok"
        }
```

```
        }
```

错误码：

- 34001,34002,34003,34004,34005,34006,34999

3.2.4 心跳报文

功能描述

- 心跳间隔时间，具体根据实际情况设置,如nginx默认websocket超时时间是60s，则可以在连接无数据交互情况持续到快过期前发送一次心跳

接口定义

接口名称：心跳报文

客户端向云端发送的JSON字符串格式数据如下（红色部分为示例数据）：

```
        {"cmd": "keepAlive"}
```

云端向客户端返回订阅结果的响应JSON字符串格式数据如下（红色部分为示例数据）：

```
        {
        "cmd":"keepAlive",
        "code":"000000",
        "desc":"success"
        }
```

错误码：

- 34001,34002,34003,34004,34005,34006,34999

3.2.5 关闭连接功能

- 关闭连接具体无交互接口，只需客户端关闭session即可。

3.2.6 自动重连机制

- 由于网络闪断、Websocket Server服务端重启升级等原因，势必造成已有Websocket Client接入端连接中断，所以强烈建议Websocket Client接入端代码增加自动重连机制，可参照3.9示例中绿色标注代码或在此基础上优化。

注：

- (1) 自动重连尝试间隔可逐步递增，如5s尝试一次连接，如果不成功则2min后再尝试一次连接，如果还未成功则5min后再尝试连接一次。
- (2) 如果(1)未重连成功，则可尝试在以(1)为一个周期，持续循环重连。
- (3) 建议重连尝试间隔不易过短或频繁,如几秒钟一循环,以防止瞬间大量访问,对服务端造成连接压力。

4 附录

4.1 返回码列表公共错误码

错误码	描述
34001	Illegal parameters.（参数非法）
34002	The type information obtained is illegal. （获取到的类型信息非法）
34003	Add subscribe relationship fail. （添加订阅关系鉴权失败）
34004	Delete subscribe relationship fail. （取消订阅关系鉴权失败）
34005	Illegal result of search. （查询结果异常）
34006	Invalid connection. （连接异常）
34999	UnKnown error. （未知异常）

4.2 签名算法示例

```
String getSign(String accesskeyId, String accesskeySecret, String timestamp) {
        accesskeyId = accesskeyId.trim();
        accesskeyId = accesskeyId.replaceAll("\"", "");

        StringBuffer sb = new StringBuffer();
        sb.append(accesskeyId).append(accesskeySecret).append(timestamp);
        MessageDigest md = null;
        byte[] bytes = null;
        try {
        md = MessageDigest.getInstance("SHA-256");
        bytes = md.digest(sb.toString().getBytes("utf-8"));
        } catch (Exception e) {
        e.printStackTrace();
        }
        return BinaryToHexString(bytes);
        }
        String BinaryToHexString(byte[] bytes) {
        StringBuilder hex = new StringBuilder();
        String hexStr = "0123456789abcdef";
        for (int i = 0; i < bytes.length; i++) {
        hex.append(String.valueOf(hexStr.charAt((bytes[i] & 0xF0) >> 4)));
        hex.append(String.valueOf(hexStr.charAt(bytes[i] & 0x0F)));
        }
        return hex.toString();
        }
```

4.3 示例

- 以下提供Client端简单功能示例代码，仅方便引导本地开发使用,具体可根据本地实际情况进行二次开发。

1、 客户端Websocket实现包的pom依赖说明

Tomcat的Websocket实现依赖包举例如下：

```
<dependency>
<groupId>org.apache.tomcat.embed</groupId>
<artifactId>tomcat-embed-websocket</artifactId>
<version>9.0.35</version>
</dependency>
```

说明：

(1) 如上插件版本最低适用于JDK8环境。

(2) 如果本地项目是SpringBoot Web工程，因为其已经默认内嵌了Tomcat相关jar（同时包含了Tomcat针对Websocket的相关实现jar），所以不必在pom.xml中单独做Websocket相关引入，但必须注意Tomcat相关jar的引用范围，如下示意：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <!-- 移除嵌入式tomcat插件 -->
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
        </exclusions>
</dependency>
<!-- 打war包时加入此项， 告诉spring-boot tomcat相关jar包用外部tomcat服务器的，不要打进去 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

2、 初始化客户端建立连接

```
try {
        String url = "ws://192.168.131.2:8091/websocket?
accesskeyId=3o80mxreadyagomr&timestamp=1651747253729&sign=b38528d5ef61118a1b72e47a4142f
2def04fef3babc17894a54beaadd0a9691f";
        String cmdInfo= "{\"cmd\":\"subscribe\", \"topics\":[\"*\"]}";
        List<String> cmdInfos = new ArrayList<>();
        cmdInfos.add(cmdInfo);
        WebsocketSubscriberClient client = new
WebsocketSubscriberClient(url,cmdInfos,processor);
        } catch (Exception e) {
        e.printStackTrace();
        }
```

3、 订阅详细消息

```
try {
        String cmdInfo= "{\"cmd\":\"subscribe\", \"topics\":[\"*\"]}";
        List<String> cmdInfos = new ArrayList<>();
        cmdInfos.add(cmdInfo);
        WebsocketSubscriberClient client = new
WebsocketSubscriberClient(url,cmdInfos,processor);
        client.addCmdInfo(cmdInfo);
        } catch (Exception e) {
        e.printStackTrace();
        }
```

4、 取消订阅详细消息

```
try {
        String cmdInfo= "{\"cmd\":\"subscribe\", \"topics\":[\"DEV_STATUS\",
\"DEV_BIGDATA\"]}";
        List<String> cmdInfos = new ArrayList<>();
        cmdInfos.add(cmdInfo);
        WebsocketSubscriberClient client = new
WebsocketSubscriberClient(url,cmdInfos,processor);

        String unSubscribeCmdInfo= "{\"cmd\":\"unsubscribe\", \"topics\":
[\"DEV_STATUS\", \"DEV_BIGDATA\"]}";
        client.addCmdInfo(unSubscribeCmdInfo);
        } catch (Exception e) {
        e.printStackTrace();
        }
```

5、 关闭当前客户端连接

```
try {
        String cmdInfo= "{\"cmd\":\"subscribe\", \"topics\":[\"DEV_STATUS\",
\"DEV_BIGDATA\"]}";
        List<String> cmdInfos = new ArrayList<>();
        cmdInfos.add(cmdInfo);
        WebsocketSubscriberClient client = new
WebsocketSubscriberClient(url,cmdInfos,processor);
        client.close();
        } catch (Exception e) {
        e.printStackTrace();
        }
```

注：如果每次建立连接后只是进行查询等一次性操作（即不需要连接长时间保留），则建议操作完后关闭当前链接，同时不建议使用心跳。

6、 整示例代码（可复用）

7、 代码功能：

- (1) 支持Client端订阅消息。
- (2) 支持心跳检测，心跳功能简单支持当客户端和服务器端连接处于无数据交互状态时才发送心跳检测消息，有数据交互时不发送。支持连接断开后指定时间进行重连尝试。

WebsocketSubscriberClient类：

```
import com.google.common.base.Throwables;
```

```
import org.apache.tomcat.websocket.WsWebSocketContainer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.websocket.*;
import java.net.URI;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicLong;

@ClientEndpoint
public class WebsocketSubscriberClient {
    private Logger logger = LoggerFactory.getLogger(WebsocketSubscriberClient.class);
    // 最新收消息时间,心跳发消息时做时间间隔计算
    public static AtomicLong lastReceiveTime = new AtomicLong();
    // 是否继续心跳标志位
    public static AtomicBoolean isHeartBeatContinue = null;
    private DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
    // 创建容器对象（建立连接时要用）
    private WebSocketContainer container = ContainerProvider.getWebSocketContainer();
    // 一个连接对应一个session
    private static Session session = null;
    // 心跳线程池
    private ExecutorService executorService = null;
    // 关闭心跳线程要用到
    private Future<?> future = null;
    private static MsgProcessor msgProcessor;
    private static String url;
    private static List<String> cmdInfos;

    public WebsocketSubscriberClient() {
    }

    public WebsocketSubscriberClient(String uri, List<String> cmdInfos, MsgProcessor
msgProcessor) throws Exception {
        WebsocketSubscriberClient.url = uri;
        WebsocketSubscriberClient.msgProcessor = msgProcessor;
        WebsocketSubscriberClient.cmdInfos = cmdInfos;
        session = container.connectToServer(WebsocketSubscriberClient.class,
URI.create(uri));
        logger.info("WebsocketSubscriberClient init finished...");
        if (session.isOpen()) {
            executorService = new ThreadPoolExecutor(1, 1, 6000, TimeUnit.MILLISECONDS,
                    new LinkedBlockingDeque<Runnable>());
            lastReceiveTime = new AtomicLong(System.currentTimeMillis());
            isHeartBeatContinue = new AtomicBoolean(true);
```

```
            // 值为0表示会话连接不会因为长时间无数据交互而超时
            session.setMaxIdleTimeout(0);
            // 1Mb
            session.setMaxTextMessageBufferSize(1024 * 1024);
            // 启动心跳
            startHeartBeat();
        } else {
            container = null;
        }
    }

    @OnMessage
    public void onMessage(final String message) {
        try {
            long currentTime = System.currentTimeMillis();
            // 更新最新收消息时间
            lastReceiveTime.set(currentTime);
            String currentDateStr = dateFormat.format(currentTime);
            logger.debug(currentDateStr + " 收到 message: " + message);
            msgProcessor.processorMsg(message);
        } catch (Exception e) {
            logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
        }
    }

    @OnClose
    public void onClose(CloseReason closeReason) {
        stopHeartBeat();
        stopContainer();
        logger.error("Connection closed!  CloseReason : code:{},reason:{}",
closeReason.getCloseCode().getCode(),
                closeReason.getReasonPhrase());
        if (closeReason.getCloseCode().getCode() !=
CloseReason.CloseCodes.NORMAL_CLOSURE.getCode()) {
            boolean isSuccess = false;
            while (!isSuccess) {
                try {
                    logger.error("Try restarting the connection after 5s...");
                    Thread.sleep(5000);
                    isSuccess = buildClient();
                    if (!isSuccess) {
                        logger.error("Try restarting the connection after 2min...");
                        Thread.sleep(120000);
                        isSuccess = buildClient();
                        if (!isSuccess) {
                            logger.error("Try restarting the connection after
5min...");
                            Thread.sleep(300000);
                            isSuccess = buildClient();
```

```
                        }
                    }
                } catch (Exception e) {
                    logger.error("Exception = {}",
Throwables.getStackTraceAsString(e));
                }
            }
        }
    }

    private boolean buildClient() {
        try {
            new WebsocketSubscriberClient(url, cmdInfos, msgProcessor);
            cmdInfos.forEach(cmd -> {
                addCmdInfo(cmd);
            });
            return true;
        } catch (Exception e) {
            logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
            return false;
        }
    }

    public void addCmdInfo(final String cmdInfo) {
        try {
            if (session.isOpen()) {
                synchronized (session) {
                    session.getBasicRemote().sendText(cmdInfo);
                }
            }
        } catch (Exception e) {
            logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
        }
    }

    public void close() {
        try {
            // 关闭心跳
            stopHeartBeat();
            // 关闭session连接
            session.close(new CloseReason(CloseReason.CloseCodes.NORMAL_CLOSURE,
"NORMAL_CLOSE"));
            // 关闭容器
            stopContainer();
        } catch (Exception e) {
            logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
        }
    }
```

```
    private void stopContainer() {
        if (container instanceof WsWebSocketContainer) {
            try {
                ((WsWebSocketContainer) container).destroy();
            } catch (Exception e) {
                logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
            }
        }
    }

    private void startHeartBeat() {
        future = executorService.submit(new HeartBeat());
    }

    private void stopHeartBeat() {
        if (future != null) {
            try {
                isHeartBeatContinue.set(false);
                future.cancel(true);
                executorService.shutdown();
            } catch (Exception e) {
                logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
            }
        }
    }

    private class HeartBeat implements Runnable {
        // 心跳间隔时间，具体根据实际情况设置,如nginx默认websocket超时时间是60s，则可以在连接无数据交互情况持续到快过期前发送一次心跳（这里设置55s时），
        private long heartBeatInterval = 55000;

        @Override
        public void run() {
            try {
                while (isHeartBeatContinue.get()) {
                    long startTime = System.currentTimeMillis();
                    if (startTime - lastReceiveTime.get() > heartBeatInterval) {
                        String cmdInfo = "{\"cmd\":\"keepAlive\"}";
                        if (session.isOpen()) {
                            synchronized (session) {
                                session.getBasicRemote().sendText(cmdInfo);
                            }
                            // 更新最新收消息时间
                            lastReceiveTime.set(System.currentTimeMillis());
                        }
                    }
                    Thread.sleep(1000);
                }
            } catch (Exception e) {
```

```
                logger.error("Exception = {}", Throwables.getStackTraceAsString(e));
            }
        }
    }
}
```

MsgProcessor接口：

```
public interface MsgProcessor {
    void processorMsg(String msg);
}
```

测试示例代码：

```
        import java.util.ArrayList;
        import java.util.List;

public class TestSubscribeMsg {
    public static void main(String[] args) {
        buildConnectionAndSubscribeMsg();
    }

    private static void buildConnectionAndSubscribeMsg() {
        try {
            String url = "ws://192.168.131.2:8091/websocket?
accesskeyId=3o80mxreadyagomr&timestamp=1651747253729&sign=b38528d5ef61118a1b72e47a4142f
2def04fef3babc17894a54beaadd0a9691f";
            String cmdInfo= "{\"cmd\":\"subscribe\", \"topics\":[\"*\"]}";
            // cmdInfos 用于websocket非正常端开时重连订阅。
            List<String> cmdInfos = new ArrayList<>();
            cmdInfos.add(cmdInfo);

            MsgProcessor processor = new MsgProcessor() {
                @Override
                public void processorMsg(String msg) {
                    System.out.println("receive msg: " + msg);
                }
            };
            WebsocketSubscriberClient client = new
WebsocketSubscriberClient(url,cmdInfos,processor);

            //发送订阅请求
            client.addCmdInfo(cmdInfo);

            //String unSubscribeCmdInfo= "{\"cmd\":\"unsubscribe\", \"topics\":
[\"DEV_STATUS\", \"DEV_BIGDATA\"]}";
            //发送取消订阅请求并关闭当前客户端连接
            //client.addCmdInfo(unSubscribeCmdInfo);

            //关闭当前客户端连接
            // client.close();
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

说明：

(1) 心跳的JSON字符串格式如下：

`{"cmd": "keepAlive"}`

(2) 服务端相应如下：

{
"cmd":"keepAlive",
"code":"000000",
"desc":"success"
}

4.4 自动重连机制

由于网络闪断、Websocket Server服务端重启升级等原因，势必造成已有Websocket Client接入端连接中断，所以强烈建议Websocket Client接入端代码增加自动重连机制，可参照示例基础上优化。

说明：

- (1) 自动重连尝试间隔可逐步递增，如5s尝试一次连接，如果不成功则2min后再尝试一次连接，如果还未成功则5min后再尝试连接一次。
- (2) 如果(1)未重连成功，则可尝试在以(1)为一个周期，持续循环重连。
- (3) 建议重连尝试间隔不易过短或频繁,如几秒钟一循环,以防止瞬间大量访问,对服务端造成连接压力。