说明： 为了用户的交易数据的安全性，防止数据被恶意篡改，现约定规则，对数据进行加密。

1. 加密算法说明。

   > 采用安全散列算法SHA256

2. 如何获取api_key以及secret？

   > 客户通过我方资管平台申请apiKey。 申请成功后，会返回一串16位字符串，称为api_key，以及一串32位字符串，称作为密匙secret，用户应该妥善保管此secret，防止泄漏。

3. 加密规则和加密过程简要说明

   > 客户发送http请求前，首先需要将传递的参数以"key01=value01&key02=value02...&secret=HkdslNJDuMysHRL"形式，对参数名，参数值以及密匙按照验签规则进行拼接，然后使用SHA256加密方式进行加密计算得到一串字符，该字符串称作签名sign。发送请求时，需要添加两个请求头，请求头api_key作为操作产品的权限标识，请求头sign作为签名，用于我方校验。

4. 交互说明

   > 客户使用http发送请求调用我方交易相关的API对对应的产品进行交易操作，需要做如下操作。
   >
   > a）需要添加请求头api_key， 值为资管平台三方接口页面上显示的apiKey。
   >
   > b）需要添加请求头sign，值为按约定规则加密后的字符串。

5. 关于sign的加密规则示例，我方会提供源码，请参照Demo。

   > 签名DEMO说明：
   > 需要传递的参数，用key-value结构保存为Map<String， String> params.然后传入params以及secret，调用下述的方法即可得到签名sign。（补充说明， 此方法会将传入的参数名和参数值以及secret及其值进行拼接。拼接的规则是，依据参数名字典顺序排序，最后拼接secret。如，传入参数有exchange=OKEX ，contract=ADA/BTC 以及需要拼接的密匙secret=TQAFrtRS4WrUn1hHKByszao3ozWvIMkF,那么拼接后的原始字符串是，contract=ADA/BTC&exchange=OKEX&secret=TQAFrtRS4WrUn1hHKByszao3ozWvIMkF，调用SHA-256加密方法即可得到sign。）

```
package com.ceres.signutil;

import org.junit.Test;
import org.springframework.util.StringUtils;

import java.security.MessageDigest;
import java.util.Arrays;
```

```
import java.util.HashMap;
import java.util.Map;

public class SignDemo {

    public static String signBySha256(Map<String, String> params, String
secret) throws Exception {
        String[] keys = params.keySet().toArray(new String[0]);
        Arrays.sort(keys);
        StringBuilder query = new StringBuilder();
        for (String key : keys) {
            String value = params.get(key);
            if (!(StringUtils.isEmpty(key)||StringUtils.isEmpty(value)))
{
                query.append(key).append("=").append(value).append("&");
            }
        }
        query.append("secret=").append(secret);
        System.out.println("签名前的字符串:  " + query);
        MessageDigest messageDigest = MessageDigest.getInstance("SHA-
256");
        messageDigest.update(query.toString().getBytes("UTF-8"));
        return byte2Hex(messageDigest.digest());
    }

    private static String byte2Hex(byte[] bytes) {
        StringBuffer stringBuffer = new StringBuffer();
        String temp = null;
        for (int i = 0; i < bytes.length; i++) {
            temp = Integer.toHexString(bytes[i] & 0xFF);
            if (temp.length() == 1) {
                //1得到一位的进行补0操作
                stringBuffer.append("0");
            }
            stringBuffer.append(temp);
        }
        return stringBuffer.toString();
    }

    @Test
    public void testSign() throws Exception {
        // 入参。 exhange = BITMEX
        Map<String, String> params = new HashMap<>();
        params.put("exchange","BITMEX");
        String secret = "TQAFrtRS4WrUn1hHKByszao3ozWvIMkF";
        String sign = signByMd5(params, secret);
        System.out.println("签名后得到sign:  " + sign);
    }
}
```

console:
签名前的字符串： exchange=BITMEX&secret=TQAFrtRS4WrUn1hHKByszao3ozWvIMkF
签名后得到sign:
9173fddc96e445b07a3d11295b52ca45385f96d9a52307da58134a2fd7e239be

场景说明:
生产环境测试工作如下,
在资管平台的第三方接口页面，点击新增按钮，得到api_key和secret。注意妥善保存!
api_key:
TEIrrNVUAUnujLfn
secret:
TQAFrtRS4WrUn1hHKByszao3ozWvIMkF

签名示例：例如，获取交易所exchange为BITMEX的交易对列表。

签名前，拼接好的原始字符串:
exchange=BITMEX&secret=TQAFrtRS4WrUn1hHKByszao3ozWvIMkF
签名后的加密字符串sign:
9173fddc96e445b07a3d11295b52ca45385f96d9a52307da58134a2fd7e239be

发送请求:
在请求头中添加头
api_key:TEIrrNVUAUnujLfn
sign:9173fddc96e445b07a3d11295b52ca45385f96d9a52307da58134a2fd7e239be

请求路径path:
open.coinceres.com/api/v1/basic/contracts?exchange=BITMEX

response响应:

```
{
    "code": "200",
    "data": [
        {
            "contract": "XBTUSD",
            "exchange": "BITMEX",
            "min_change": "0.5",
            "min_volume": "1"
        },
        {
            "contract": "ADAZ18",
            "exchange": "BITMEX",
            "min_change": "0.00000001",
            "min_volume": "1"
        },
  …...
    ],
    "message": "SUCESS"
}
```