Java和移动端交互

我们开发java EE项目,不止有B/S模式,还有C/S模式,也就是客户端-服务端的方式,现在的主流客户端基本都是移动端,那么java如何和移动端交互呢

> 其实对于任何模式的请求来说,都没有任何区别,因为他们最终都被封装为java总的request对象,所以在我们的java服务的来说我们是不需要做什么改动的,只是因为移动端我们返回一个页面给它的话它解析起来比较麻烦,所以移动端都是以返回json数据的格式操作,也就是相当于我们在html中的ajax异步请求返回json的操作

加密

因为http协议的原因,它会将数据以明文的方式进行发送,所以http协议是不安全的,我们应该将数据进行响应处理之后再发送

app的开发和网页开发不一样,http协议中网页中对无法对数据进行加密处理,app因为是代码开发的,可以方便对数据加密,防止敏感数据被窃取

加密方式

对称性加密

> 所谓的对称性加密指的是加密密码和解密密码一致的加密方式,也就是在服务端和客户端上面都要保存有密码,这种方式适合对安全性能要求没有那么高的产品,主要是方式数据在通信过程中被破解,但是如果客户端被破解,则会暴露密码

常见对称加密方式

DES

DES全称为Data Encryption Standard，即数据加密标准，是一种使用密钥加密的块算法，1977年被美国联邦政府的国家标准局确定为联邦资料处理标准（FIPS），并授权在非密级政府通信中使用，随后该算法在国际上广泛流传开来

AES

高级加密标准（英语：Advanced Encryption Standard，缩写：AES），在密码学中又称Rijndael加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的DES，已经被多方分析且广为全世界所使用

非对称性加密

> 区别于对称性加密,费对称性加密指的是加密和解密密码不一致的方式,非对称加密算法需要两个密钥:公开密钥（publickey）和私有密钥（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法

常见非对称加密方式

RSA

RSA是目前最有影响力和最常用的公钥加密算法，它能够抵抗到目前为止已知的绝大多数密码攻击,RSA算法基于一个十分简单的数论事实：将两个大质数相乘十分容易，但是想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

在公开密钥密码体制中，加密密钥（即公开密钥）PK是公开信息，而解密密钥（即秘密密钥）SK是需要保密的。加密算法E和解密算法D也都是公开的。虽然解密密钥SK是由公开密钥PK决定的，但却不能根据PK计算出SK。

RSA算法是第一个能同时用于加密和数字签名的算法，也易于理解和操作,HTTPS的证书使用的便是RSA

非加密方式

有一些我们经常会被误认为是加密的非加密算法,在实际开发中,我们经常会口误将其表达为加密,下面我们列出一些非加密方式

MD5

> MD5是最常见的非加密方式,也是被误认为是加密最多的方式,MD5其实是一种基于消息摘要算法实现的散列函数,用于确保信息传输完整一致,主要用于校验,数据库中存放的是原始内容的MD5值,然后将传递过来的内容转换为MD5,之后进行比较,如果一致,则代表输入正确,MD5是不可逆的,网上的所谓解密网站其实是通过不断计算存储各种数据的MD5,然后将输入的MD5和数据库中存储的进行比较得到的结果

MD5算法具有以下特点：

1. 压缩性：任意长度的数据，算出的MD5值长度都是固定的。
2. 容易计算：从原数据计算出MD5值很容易。
3. 抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。
4. 强抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

MD5的作用是让大容量信息在用数字签名软件签署私人密钥前被"压缩"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的十六进制数字串）。除了MD5以外，其中比较有名的还有sha-1、RIPEMD以及Haval等。

SHA

安全散列算法SHA,是美国国家标准技术研究所发布的国家标准FIPS PUB 180，最新的标准已经于2008年更新到FIPS PUB 180-3。其中规定了SHA-1，SHA-224，SHA-256，SHA-384，和SHA-512这几种单向散列算法。SHA-1，SHA-224和SHA-256适用于长度不超过2^64二进制位的消息。SHA-384和SHA-512适用于长度不超过2^128二进制位的消息

使用方式

非对称使用方式

在和移动交互中,包括和网页的交互中,使用非对称加密的方式进行加密是安全级别最高的,其实我们只需要将网站修改为https即可,服务端不需要做任何事情,只需要移动端做处理即可,但是注意,使用https的话,如果使用的是非浏览器认为的安全机构签名的证书或者自制证书,浏览器会警告网站不安全,但是不影响通讯
配置https的方式,请参考 tomcat配置https的文档

对称性使用方式

对称性方式是普通app中最常见的加密方式,使用简单,也有一定安全性,使用代码网上一搜索一大堆,需要注意的是,我们加密后的数据是二进制数据,是无法直接转换为字符串发送到服务端的,如果直接转换会导致数据的丢失,导致无法解密,所以我们需要配合BASE64编码的方式将其转换为字符串

代码

> 因为双方操作的一致性,所以我们只需要写一份代码工具类,然后给移动端即可,仅限于android,ios需要单独写,使用方式
>
> 加密:
>
> 1.使用encryptNew方法将要传递的字符串进行加密,得到byte数组
>
> 2.使用parseByte2HexStr将加密后的byte数组转码为字符串,此字符串和原始内容不一致
>
> 解密:
>
> 1.收到数据后,首先调用parseHexStr2Byte将参数进行解码,变回原始的加密后数据
>
> 2.将第三步的返回结果作为参数调用decryptNew得到一个解密后的byte数组,这数组是原始内容的数组,可以直接new为字符串
>
> 服务端需要向客户端返回加密的数据,使用步骤同上,只不过变为服务的加密,客户端解密

```
/**
 * Created by jackiechan on 17/2/27. 加密必须得保证数据安全性(解密后的数据正确性)
 * 加密工具类
```

```
 *
 */

public class SecretUtils {


    /**
     *  Base64编码方法
     * 将二进制转换成16进制,将加密后的二进制数据转字符串
     *
     * @param buf
     * @return
     */
    public static String parseByte2HexStr(byte buf[]) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < buf.length; i++) {
            String hex = Integer.toHexString(buf[i] & 0xFF);
            if (hex.length() == 1) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase());
        }
        return sb.toString();
    }

    /**
     * Base64解码,将字符串解码为编码前的数据,即转回加密后的二进制数据,
     * 将16进制转换为二进制 ,利用上面的转换方式转出的显示内容需要先利用此方式转码后解密用
     *
     * @param hexStr
     * @return
     */
    public static byte[] parseHexStr2Byte(String hexStr) {
        if (hexStr.length() < 1)
            return null;
        byte[] result = new byte[hexStr.length() / 2];
        for (int i = 0; i < hexStr.length() / 2; i++) {
            int high = Integer.parseInt(hexStr.substring(i * 2, i * 2 + 1), 16);
            int low = Integer.parseInt(hexStr.substring(i * 2 + 1, i * 2 + 2),
16);
            result[i] = (byte) (high * 16 + low);
        }
        return result;
    }
```

```
/**
 * 和 android 兼容的加密算法,将字符串加密为二进制数组
 * @param value
 * @return
 */
    public static byte[] encryptNew(String value) {
        byte[] encrypted = null;
        try {
            byte[] raw = new byte[]{'T', 'h', 'i', 's', 'I', 's', 'A', 'S', 'e',
'c', 'r', 'e', 't', 'K', 'e', 'y'};//密码
            Key skeySpec = new SecretKeySpec(raw, "AES");
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            byte[] iv = new byte[cipher.getBlockSize()];
            IvParameterSpec ivParams = new IvParameterSpec(iv);
            cipher.init(Cipher.ENCRYPT_MODE, skeySpec,ivParams);
            encrypted  = cipher.doFinal(value.getBytes());
            System.out.println("encrypted string:" + encrypted.length);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return encrypted;
    }
    /**
     * 保证和移动端兼容,将加密后的二进制数据进行解密
     * @param encrypted
     * @return
     */
    public static  byte[]  decryptNew(byte[] encrypted) {
        byte[] original = null;
        Cipher cipher = null;
        try {
            byte[] raw = new byte[]{'T', 'h', 'i', 's', 'I', 's', 'A', 'S', 'e',
'c', 'r', 'e', 't', 'K', 'e', 'y'};
            Key key = new SecretKeySpec(raw, "AES");
            cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            byte[] ivByte = new byte[cipher.getBlockSize()];
            IvParameterSpec ivParamsSpec = new IvParameterSpec(ivByte);
            cipher.init(Cipher.DECRYPT_MODE, key, ivParamsSpec);
            original= cipher.doFinal(encrypted);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return original;
    }
```

```
/**
     * AES加密字符串,注意此方法在andorid6.0后不兼容
     *
     * @param content
     *            需要被加密的字符串
     * @param password
     *            加密需要的密码
     * @return 密文

    public static byte[] encrypt2(String content, String password) {
        try {
            KeyGenerator kgen = KeyGenerator.getInstance("AES");// 创建AES的Key生
产者
            SecureRandom random = SecureRandom.getInstance("SHA1PRNG","Crypto");
            random.setSeed(password.getBytes());
            kgen.init(128, random);// 利用用户密码作为随机数初始化出
            // 加密没关系，SecureRandom是生成安全随机数序列，password.getBytes()是种
子，只要种子相同，序列就一样，所以解密只要有password就行
            SecretKey secretKey = kgen.generateKey();// 根据用户密码，生成一个密钥
            byte[] enCodeFormat = secretKey.getEncoded();// 返回基本编码格式的密
钥，如果此密钥不支持编码，则返回null
            SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");// 转换为
AES专用密钥
//            Cipher cipher = Cipher.getInstance("AES");// 创建密码器
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");// 创建密码
器
            byte[] byteContent = content.getBytes("utf-8");
            cipher.init(Cipher.ENCRYPT_MODE, key);// 初始化为加密模式的密码器
            byte[] result = cipher.doFinal(byteContent);// 加密
            return result;

        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (NoSuchProviderException e) {
            e.printStackTrace();
        }
        return null;
```

```
    }
     */
    /**
     * 解密AES加密过的字符串 ,此方法在andorid6.0后不兼容
     *
     * @param content
     *            AES加密过过的内容
     * @param password
     *            加密时的密码
     * @return 明文

    public static byte[] decrypt2(byte[] content, String password) {
        try {
            KeyGenerator kgen = KeyGenerator.getInstance("AES");// 创建AES的Key生
产者
            SecureRandom random = SecureRandom.getInstance("SHA1PRNG","Crypto");
            random.setSeed(password.getBytes());
            kgen.init(128, random);// 利用用户密码作为随机数初始化出
            SecretKey secretKey = kgen.generateKey();// 根据用户密码，生成一个密钥
            byte[] enCodeFormat = secretKey.getEncoded();// 返回基本编码格式的密钥
            SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");// 转换为
AES专用密钥
            //Cipher cipher = Cipher.getInstance("AES");// 创建密码器
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");// 创建密码
器
            cipher.init(Cipher.DECRYPT_MODE, key);// 初始化为解密模式的密码器
            byte[] result = cipher.doFinal(content);
            return result; // 明文
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (NoSuchProviderException e) {
            e.printStackTrace();
        }
        return null;
    }

     */

}
```

过滤器

因为上述步骤,需要在每个方法中都做操作,比较麻烦,而且有时候我们的接口可能也会给网页端使用,所以可能传递过来的数据是非加密的,那么我们如何区分是移动端还是网页端呢,以及如何统一对加密数据进行解密,而不需要每次都写呢
我们需要写一个自定义的过滤器,这个过滤器中可以判断是否为移动端,并且自定义request对象,在这个对象中,重写获取参数的方法,如果是加密的,则进行解密,如果是网页的数据,则不进行任何处理

1:如何判断是否是移动端,可以在app端中每个接口额外传递一个参数,name自定义 比如 platform,如果这个参数有值,则代表是移动端,反之是网页端

Filter代码

```
/**
 * Created by jackiechan on 17/2/27.
 * 用于处理加密的Filter
 *
 */
public class MySuperFilter implements Filter {
    public void destroy() {
    }

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain
chain) throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        MyHttpServletRequest myHttpServletRequest=new
MyHttpServletRequest((HttpServletRequest) req);
        chain.doFilter(myHttpServletRequest, resp);
        Common.isMobileLocal.remove();//从Treadlocal中移除本线程的移动端判断boolean值
    }

    public void init(FilterConfig config) throws ServletException {

    }

}
```

自定义Request代码

```
/**
 * Created by jackiechan on 17/2/27.
 * 用于判断是否为加密,并且转化数据的自定义request
 *
```

```
 */
public class MyHttpServletRequest extends HttpServletRequestWrapper {
    private boolean isMobile=false;

    public boolean isMobile() {
        return isMobile;
    }

    public MyHttpServletRequest(HttpServletRequest request) {
        super(request);
        String platmform = super.getParameter("platform");//获取请求参数中的platform
的值
        if (platmform == null || "".equalsIgnoreCase(platmform)) {//如果有值,则代表
是移动端,此处因为调用的是super所以方法获取的是非解密的数据
            isMobile = false;
        }else{

        isMobile="mobile".equals(new
String(SecretUtils.decryptNew(SecretUtils.parseHexStr2Byte(platmform))));//对传递的
值进行解密,如果是mobile则代表是移动端,此处的值可以随便写
        }
        Common.isMobileLocal.set(isMobile);//将值放到threadloca中,方便在后面返回数据
的时候判断是移动端进行加密返回
    }

    /**
     * 重写获取参数的方法,如果是移动端,则进行解密
     * @param name
     * @return
     */
    @Override
    public String getParameter(String name) {
        if (!isMobile) {
            return super.getParameter(name);
        }
        String source = super.getParameter(name);//拿到原始的字符串
        //解密
        byte[] hexStr2Byte = SecretUtils.parseHexStr2Byte(source);//转回原始的二进制
数据
        byte[] decryptNew = SecretUtils.decryptNew(hexStr2Byte);//解密后的数据
        return new String(decryptNew);//转成字符串

    }

    @Override
    public String[] getParameterValues(String name) {
        if (!isMobile) {
            return super.getParameterValues(name);
        }
```

```
        String[] parameterValues = super.getParameterValues(name);
        return changeStringrray(parameterValues);
    }

    /**
     * 重写获取参数的方法,如果是移动端,则进行解密
     * @return
     */
    @Override
    public Map<String, String[]> getParameterMap() {
        if (!isMobile) {
            return  super.getParameterMap();
        }
        Map<String, String[]> map = super.getParameterMap();
        Map<String, String[]> newmap =new HashMap<String, String[]>();//用于存放解
密后的数据的对象
        //将原始 map 中的加密的数据解密放到新的 map 中
        Set<Map.Entry<String, String[]>> entrySet = map.entrySet();
        for (Map.Entry<String, String[]> entry : entrySet) {
            String[] value = entry.getValue();//加密后的数据
            String[] strings = changeStringrray(value);//解密后的数据
            newmap.put(entry.getKey(), strings);
        }

        return newmap;
    }

    /**
     * 解密加密的数据
     * @param parameterValues
     * @return
     */
    private String[] changeStringrray(String[] parameterValues) {
        if (parameterValues!=null) {
            String[] newSStrings=new String[parameterValues.length];
            for (int i = 0; i < parameterValues.length; i++) {
                String source = parameterValues[i];//加密后的字符串
                byte[] hexStr2Byte = SecretUtils.parseHexStr2Byte(source);//转回原
始的二进制数据
                byte[] decryptNew = SecretUtils.decryptNew(hexStr2Byte);//解密后的
数据
                newSStrings[i]=new String(decryptNew);//将解密后的字符串单放到数组里
面
            }
            return newSStrings;
        }
        return null;
    }
}
```

注意

> 然后在项目的web.xml中配置我们的Filter过滤器即可,需要注意,如果有自定义的字符集的filter,我们的加密过滤器的filter-mapping需要配置在字符集的后面,配置在其他的filter的前面