RSA 加密工具类

代码块

```
import javax.crypto.Cipher;
import java.io.ByteArrayOutputStream;
import java.security.Key;
import java.security.KeyFactory;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

/**
 * RSA 加密工具类（示例）
 * <p>
 * 用途：
 * - 客户端可以使用此类进行公钥加密，将加密后的数据传给服务端
 * - 服务端接收到数据后，再用私钥解密
 * <p>
 * 注意事项：
 * - RSA 加密有分段限制，1024 位密钥最大支持 117 字节的明文块
 * - 本工具类仅支持 **公钥加密**
 *
 * @author ChatGPT
 */
public class RSAEncryptor {

    /**
     * 算法名称
     */
    private static final String ALGORITHM = "RSA";

    /**
     * 加密/解密填充方式（推荐使用 RSA/ECB/PKCS1Padding）
     */
    private static final String CIPHER_TRANSFORMATION = "RSA/ECB/PKCS1Padding";

    /**
     * RSA 最大加密块大小（1024 位密钥 = 117 字节）
     */
    private static final int MAX_ENCRYPT_BLOCK = 117;

    private RSAEncryptor() {
        // 工具类禁止实例化
    }
```

```
41
42      /**
43       * 使用公钥加密（返回 Base64 字符串）
44       *
45       * @param plainText 明文字符串
46       * @param publicKey 公钥字符串（Base64 编码）
47       * @return 加密后的 Base64 编码密文
48       * @throws Exception 加密过程中的异常
49       */
50      public static String encryptByPublicKey(String plainText, String publicKey)
    throws Exception {
51          byte[] data = plainText.getBytes("UTF-8");
52          byte[] encryptedData = encryptByPublicKey(data, publicKey);
53          return Base64.getEncoder().encodeToString(encryptedData);
54      }
55
56      /**
57       * 使用公钥加密（返回字节数组）
58       *
59       * @param data      明文字节数组
60       * @param publicKey 公钥字符串（Base64 编码）
61       * @return 加密后的字节数组
62       * @throws Exception 加密过程中的异常
63       */
64      public static byte[] encryptByPublicKey(byte[] data, String publicKey)
    throws Exception {
65          // 1. 解析公钥
66          byte[] keyBytes = Base64.getDecoder().decode(publicKey);
67          X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);
68          KeyFactory keyFactory = KeyFactory.getInstance(ALGORITHM);
69          Key key = keyFactory.generatePublic(keySpec);
70
71          // 2. 初始化 Cipher
72          Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);
73          cipher.init(Cipher.ENCRYPT_MODE, key);
74
75          // 3. 分段加密
76          int inputLength = data.length;
77          int offset = 0;
78          int i = 0;
79          byte[] cache;
80          ByteArrayOutputStream out = new ByteArrayOutputStream();
81
82          while (inputLength - offset > 0) {
83              if (inputLength - offset > MAX_ENCRYPT_BLOCK) {
84                  cache = cipher.doFinal(data, offset, MAX_ENCRYPT_BLOCK);
85              } else {
```

```
86                  cache = cipher.doFinal(data, offset, inputLength - offset);
87              }
88              out.write(cache, 0, cache.length);
89              i++;
90              offset = i * MAX_ENCRYPT_BLOCK;
91          }
92
93          byte[] encryptedData = out.toByteArray();
94          out.close();
95          return encryptedData;
96      }
97  }
```