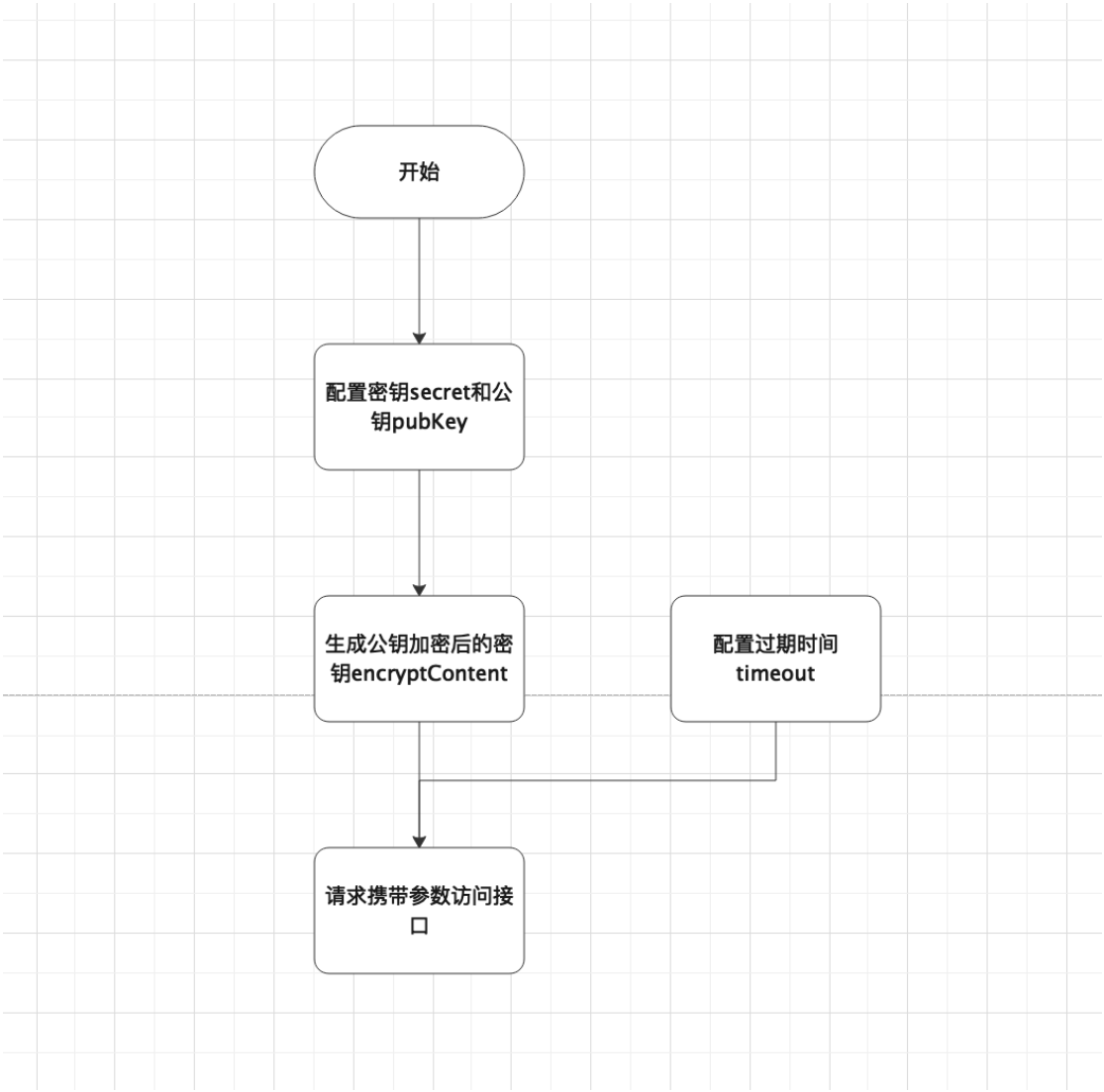
服务暴露接口对接方案

外部厂商

为外部服务设计新的鉴权方式是为了解决传统鉴权机制在开放环境下的局限性，并应对跨系统交互时的安全风险。

以下是开放给外部厂商的核心代码逻辑



代码逻辑

```
// 注意：pubKey的值是1行内容
public static void main(String[] args) throws JsonProcessingException {
        String secret = "%&*(IO)P)HGJKL_+0fah1548$%^&*(6";
        String token = JWTUtils.createToken(timestamp, new HashMap<>(), 20, secret);
        String pubKey =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDNMmo7x7QLXpdB0csrdWraSqoSv9b67JvUepO6wxXM4OY
Tv+6Ynbxwv6eEIPKpIcux3BaP7QfoLB7xN9WrtJfM/iOPA/zkLFWZ/S6+mh/lgkZIABFU87RgRDAqEJqxu2z
u8n53oCHI351/BFidoozIn/mTMB3bCsXOPd6/pJ0ZKQIDAQAB";

        String encryptContent = RSAUtils.encrypt(token, pubKey);
    }
```

```
@Slf4j
@Component
public class JWTUtils {

    private JWTUtils() {
    }

    /**
     * 生成token
     * @param timestamp 后续校验业务用来做防重放验证
     * @param payload 可以存放用户的一些信息，不要存放敏感字段
     * @param secret HS256(HmacSHA256)密钥
     * @return
     */
    public static String createToken(Map<String, Object> payload, int timeout,
String secret) {
        //十分重要，不禁用发布到生产环境无法验证
        GlobalBouncyCastleProvider.setUseBouncyCastle(false);
        DateTime now = DateTime.now();
        DateTime expTime = now.offsetNew(DateField.SECOND, timeout);
        // 签发时间
        payload.put(RegisteredPayload.ISSUED_AT, now);
        // 过期时间
        payload.put(RegisteredPayload.EXPIRES_AT, expTime);
        // 生效时间
        payload.put(RegisteredPayload.NOT_BEFORE, now);
        String token = JWTUtil.createToken(payload, secret.getBytes());
        log.info("生成JWT token：{}", token);
        return token;
    }


    /**
     * 检验token是否有效
     *
     * @param token jwt
```

```
     * @param key   HS256(HmacSHA256)密钥
     * @return
     */
    public static boolean validate(String token, String key) {
        GlobalBouncyCastleProvider.setUseBouncyCastle(false);
        try {
            JWT jwt = JWTUtil.parseToken(token).setKey(key.getBytes());
            // validate包含了verify
            boolean validate = jwt.validate(0);
            log.info("JWT token校验结果：{}", validate);
            return validate;
        } catch (Exception e) {
            log.info("检验token异常{}", e.getMessage());
            return false;
        }
    }
}

```

```
public class RSAUtils {
    private static final Logger log = LoggerFactory.getLogger(RSAUtils.class);

    private RSAUtils() {
    }

    public static String encrypt(String content, String publicKey) {
        try {
            RSA rsa = new RSA((String)null, publicKey);
            return rsa.encryptBase64(content, KeyType.PublicKey);
        } catch (Exception var3) {
            Exception e = var3;
            log.info("rsa encrypt error :{}", e);
            return null;
        }
    }

```

```
        <!--    鉴权auth    -->
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.8.0</version>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-redis</artifactId>
            <version>2.6.9</version>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.20</version> <!-- 2025年最新稳定版 -->
        </dependency>
```

改造方式

外部接口调用服务内部接口时，**在header中拼装上面代码中生成的token和timeout**，可以作为外部系统调用本服务安全认证的一种方式。

- 供应商带着timeout到服务端，服务端收到后进行防重放验证。header的key是 `timeout`，timeout单位是秒，10S <= timeout <= 3600S，建议30S（后来统一改成了1800S，即半小时）。timeout的值是以上代码中`JWTUtils.createToken(timestamp, new HashMap<>(), 30, secret);`的30。
- 供应商带着加密后的token到服务端，服务端收到后先解密，再校验是否过期，再和自己生成的token比对是否一致。通过后则鉴权通过。header的key是 `X-Access-Token`。X-Access-Token的值是`String encryptContent = RSAUtils.encrypt(token, pubKey);`的encryptContent。