- **jwt**
  - 认证机制介绍
    - HTTP Basic Auth
    - Cookie-Session Auth
    - OAuth
    - Token Auth
  - JWT介绍
  - JWT的数据结构
    - JWT头
    - 有效载荷
    - 签名
  - JWT签名算法
  - jjwt介绍
  - jwt入门案例
    - 第一步：创建maven工程jwt_demo并配置pom.xml文件
    - 第二步：编写单元测试
  - 自定义spring boot starter
    - 开发starter
      - 第一步：初始化项目
      - 第二步：修改pom文件
      - 第三步：编写实体类JwtUserInfo
      - 第四步：编写实体类Token
      - 第五步：编写工具类RsaKeyHelper
      - 第六步：编写工具类NumberHelper
      - 第七步：编写工具类StrHelper
      - 第八步：编写接口BaseException
      - 第九步：编写接口BaseExceptionCode
      - 第十步：编写类BaseUncheckedException
      - 第十一步：编写类ExceptionCode
      - 第十二步：编写类BizException
      - 第十三步：编写常量工具类BaseContextConstants
      - 第十四步：编写日期工具类DateUtils
      - 第十五步：编写工具类JwtHelper
      - 第十六步：编写类AuthClientConfigurationProperties
      - 第十七步：编写工具类JwtTokenClientUtils
      - 第十八步：编写配置类AuthClientConfiguration
      - 第十九步：编写注解EnableAuthClient
      - 第二十步：编写类AuthServerConfigurationProperties
      - 第二十一步：编写工具类JwtTokenServerUtils
      - 第二十二步：编写工具类AuthServerConfiguration
      - 第二十三步：编写注解EnableAuthServer
    - 使用starter
      - 第一步：导入tools-jwt的依赖
      - 第二步：在资源路径下创建keys目录，将通过RSA算法生成的公钥和私钥复制到此目录下
      - 第三步：编写application.yml文件
      - 第四步：在启动类加注解@EnableAuthServer
      - 第五步：编写UserController
      - 第六步：启动服务
      - 第七步：访问

jwt

认证机制介绍

HTTP Basic Auth

HTTP Basic Auth 是一种简单的登录认证方式，Web浏览器或其他客户端程序在请求时提供用户名和密码，通常用户名和密码会通过HTTP头传递。简单点说就是每次请求时都提供用户的username和password

这种方式是先把用户名、冒号、密码拼接起来，并将得出的结果字符串用Base64算法编码。

例如，提供的用户名是 `bill` 、口令是 `123456` ，则拼接后的结果就是 `bill:123456` ，然后再将其用Base64编码，得到 `YmlsbDoxMjM0NTY=` 。最终将Base64编码的字符串发送出去，由接收者解码得到一个由冒号分隔的用户名和口令的字符串。

优点：

基本上所有流行的网页浏览器都支持基本认证。

缺点：

由于用户名和密码都是Base64编码的，而Base64编码是可逆的，所以用户名和密码可以认为是明文。所以只有在客户端和服务器主机之间的连接是安全可信的前提下才可以使用。

Cookie-Session Auth

Cookie-session 认证机制是通过浏览器带上来Cookie对象来与服务器端的session对象匹配来实现状态管理。

第一次请求认证在服务端创建一个Session对象，同时在用户的浏览器端创建了一个Cookie对象；当我们关闭浏览器的时候，cookie会被删除。但可以通过修改cookie 的expire time使cookie在一定时间内有效。

优点:

相对HTTP Basic Auth更加安全。

缺点:

这种基于cookie-session的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于session认证应用的问题就会暴露出来。

OAuth

OAuth 是一个关于授权（authorization）的开放网络标准。允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。现在的版本是2.0版。

严格来说，OAuth2不是一个标准协议，而是一个安全的授权框架。它详细描述了系统中不同角色、用户、服务前端应用（比如API），以及客户端（比如网站或移动App）之间怎么实现相互认证。

优点：

- 快速开发，代码量小，维护工作少。
- 如果API要被不同的App使用，并且每个App使用的方式也不一样，使用OAuth2是个不错的选择。

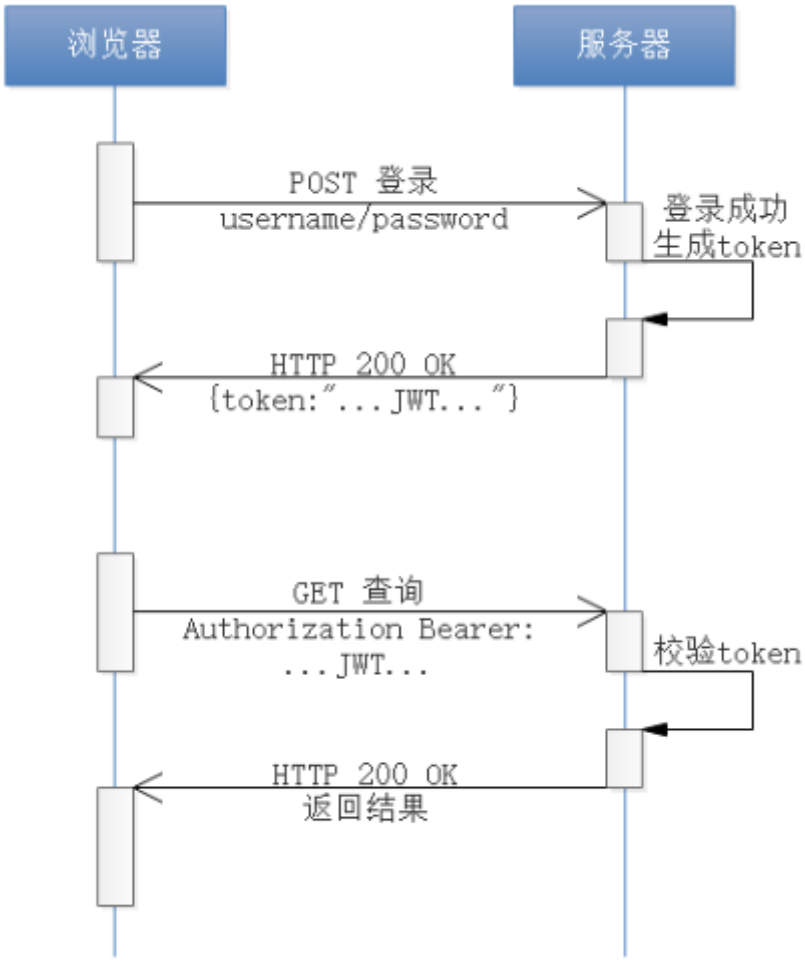
缺点：

OAuth2是一个安全框架，描述了在各种不同场景下，多个应用之间的授权问题。有海量的资料需要学习，要完全理解需要花费大量时间。OAuth2不是一个严格的标准协议，因此在实施过程中更容易出错。

Token Auth

基于token的认证鉴权机制类似于http协议，也是无状态的。这种方式不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

这个token必须要在每次请求时传递给服务端，它应该保存在请求头中，Token Auth 流程如下图：



优点：

- 支持跨域访问
- Token机制在服务端不需要存储session信息：Token 自身包含了所有登录用户的信息，只需要在客户端的cookie或本地介质存储状态信息
- 去耦：不需要绑定到一个特定的身份验证方案。Token可以在任何地方生成，只要在你的API被调用的时候，你可以进行Token生成调用即可
- 更适用于移动应用：Cookie是不被客户端（iOS, Android，Windows 8等）支持的。
- 基于标准化：
  API可以采用标准化的 JSON Web Token (JWT)。这个标准已经存在多个后端库（.NET, Ruby, Java,Python, PHP）和多家公司的支持（如：Firebase,Google, Microsoft）

缺点：

- 占带宽
  正常情况下要比 session_id 更大，需要消耗更多流量，挤占更多带宽，假如你的网站每月有 10 万次的浏览器，就意味着要多开销几十兆的流量。听起来并不多，但日积月累也是不小一笔开销。实际上，许多人会在 JWT 中存储的信息会更多
- 无法在服务端注销，因为服务端是无状态的，并没有保存客户端用户登录信息
- 对于有着严格性能要求的 Web 应用并不理想，尤其对于单线程环境

JWT介绍

JWT全称为JSON Web Token，是目前最流行的跨域身份验证解决方案。JWT是为了在网络应用环境间传递声明而制定的一种基于JSON的开放标准。

JWT特别适用于分布式站点的单点登录（SSO）场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可被加密。

JWT的数据结构

WT其实就是一个很长的字符串，字符之间通过"."分隔符分为三个子串，各字串之间没有换行符。每一个子串表示了一个功能块，总共有三个部分：**JWT头(header)**、**有效载荷(payload)**、**签名(signature)**，如下图所示：

JWT头

JWT头是一个描述JWT元数据的JSON对象，通常如下所示：

```
{"alg": "HS256","typ": "JWT"}
```

alg：表示签名使用的算法，默认为HMAC SHA256（写为HS256）

typ：表示令牌的类型，JWT令牌统一写为JWT

最后，使用Base64 URL算法将上述JSON对象转换为字符串

有效载荷

有效载荷，是JWT的主体内容部分，也是一个JSON对象，包含需要传递的数据。

有效载荷部分规定有如下七个默认字段供选择：

```
iss：发行人
exp：到期时间
sub：主题
aud：用户
nbf：在此之前不可用
iat：发布时间
jti：JWT ID用于标识该JWT
```

除以上默认字段外，还可以自定义私有字段。

最后，同样使用Base64 URL算法将有效载荷部分JSON对象转换为字符串

签名

签名实际上是一个加密的过程，是对上面两部分数据通过指定的算法生成哈希，以确保数据不会被篡改。

首先需要指定一个密码（secret），该密码仅仅保存在服务器中，并且不能向用户公开。然后使用JWT头中指定的签名算法（默认情况下为HMAC SHA256），根据以下公式生成签名哈希：

```
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload),secret)
```

在计算出签名哈希后，JWT头，有效载荷和签名哈希的三个部分组合成一个字符串，每个部分用"."分隔，就构成整个JWT对象

JWT签名算法

JWT签名算法中，一般有两个选择：HS256和RS256。

HS256 (带有 SHA-256 的 HMAC)是一种对称加密算法, 双方之间仅共享一个密钥。由于使用相同的密钥生成签名和验证签名, 因此必须注意确保密钥不被泄密。

RS256 (采用SHA-256 的 RSA 签名) 是一种非对称加密算法, 它使用公共/私钥对: JWT的提供方采用私钥生成签名, JWT 的使用方获取公钥以验证签名。

jjwt介绍

jjwt是一个提供JWT创建和验证的Java库。永远免费和开源(Apache License，版本2.0)，JJWT很容易使用和理解。

jjwt的maven坐标：

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

jwt入门案例

第一步：创建maven工程jwt_demo并配置pom.xml文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <!--
      -maven项目核心配置文件-
    Project name(项目名称)：jwt_demo
    Author(作者）: mao
    Author QQ：1296193245
    GitHub：https://github.com/maomao124/
    Date(创建日期)： 2022/11/2
    Time(创建时间)： 13:36
    -->
    <groupId>mao</groupId>
    <artifactId>jwt_demo</artifactId>

    <version>1.0-SNAPSHOT</version>
    <description>使用jjwt来解析和生成token</description>

    <properties>

        <maven.compiler.source>16</maven.compiler.source>
        <maven.compiler.target>16</maven.compiler.target>
    </properties>

    <dependencies>
        <!--jwt 依赖-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <!-- 测试框架 -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>

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

```
        </dependency>

        <!--java 8 版本不需要添加-->
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-impl</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-core</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>javax.activation</groupId>
            <artifactId>activation</artifactId>
            <version>1.1.1</version>
        </dependency>

    </dependencies>

</project>
```

第二步：编写单元测试

```
/**
 * 生成token，不使用签名
 */
@Test
void test1()
{
    Map<String, Object> head = new HashMap<>();
    head.put("alg", "none");
    head.put("typ", "JWT");

    Map<String, Object> body = new HashMap<>();
    body.put("userId", "10001");
    body.put("username", "张三");
    body.put("sex", "男");

    String token = Jwts.builder()
```

```
            .setHeader(head)
            .setClaims(body)
            .setId("jwt1")
            .compact();
    System.out.println(token);

 //eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMSIsImp0aSI6Imp3dDEiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.
}
```

```
C:\Users\mao\.jdks\openjdk-16.0.2\bin\java.exe ...
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMSIsImp0aSI6Imp3dDEiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.

进程已结束，退出代码为 0
```

```
/**
 * 解析token，不使用签名
 */
@Test
void test2()
{
    Jwt jwt = Jwts.parser().parse("eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0." +

 "eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMSIsImp0aSI6Imp3dDEiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.");
    Header header = jwt.getHeader();
    Object body = jwt.getBody();
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body);
}
```

```
C:\Users\mao\.jdks\openjdk-16.0.2\bin\java.exe ...
header={typ=JWT, alg=none},body={sex=男, userId=10001, jti=jwt1, username=张三}
{typ=JWT, alg=none}
{sex=男, userId=10001, jti=jwt1, username=张三}

进程已结束，退出代码为 0
```

```java
/**
 * 生成token，使用hs256签名算法
 */
@Test
void test3()
{
    Map<String, Object> head = new HashMap<>();
    head.put("alg", SignatureAlgorithm.HS256.getValue());
    head.put("typ", "JWT");

    Map<String, Object> body = new HashMap<>();
    body.put("userId", "10002");
    body.put("username", "张三");
    body.put("sex", "男");

    String token = Jwts.builder()
            .setHeader(head)
            .setClaims(body)
            .setId("jwt2")
            .signWith(SignatureAlgorithm.HS256, "123456")
            .compact();
    System.out.println(token);
    //eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
    //
.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW
8oOS4iSJ9
    // .9TC0U77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98
}
```

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9
  .9TC0U77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98

进程已结束，退出代码为 0
```

```java
/**
 * 解析token，使用hs256签名算法，不设置SigningKey的情况
 */
@Test
void test4()
{
    String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9" +

 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6I
uW8oOS4iSJ9." +
            "9TC0U77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98";

    Jwt jwt = Jwts.parser()
            .parse(token);
    Header header = jwt.getHeader();
    Object body = jwt.getBody();
```

```
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body);
}
```

```
java.lang.IllegalArgumentException: A signing key must be specified if the specified JWT is digitally signed.

    at io.jsonwebtoken.lang.Assert.notNull(Assert.java:85)
    at io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:331)
    at mao.JwtTest.test4(JwtTest.java:106) <31 个内部行>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 个内部行>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <26 个内部行>
```

```
/**
 * 解析token，使用hs256签名算法，SigningKey错误的情况
 */
@Test
void test5()
{
    String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9" +

 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9." +
            "9TCOU77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98";

    Jwt jwt = Jwts.parser()
            .setSigningKey("1236")
            .parse(token);
    Header header = jwt.getHeader();
    Object body = jwt.getBody();
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body);
}
```

```
io.jsonwebtoken.SignatureException: JWT signature does not match locally computed signature. JWT validity cannot be asserted and should not be trusted.

    at io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:354)
    at mao.JwtTest.test5(JwtTest.java:127) <31 个内部行>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 个内部行>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <26 个内部行>
```

```java
/**
 * 解析token，使用hs256签名算法，SigningKey正确的情况
 */
@Test
void test6()
{
    String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9" +

 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6I
uW8oOS4iSJ9." +
            "9TC0U77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98";

    Jwt jwt = Jwts.parser()
            .setSigningKey("123456")
            .parse(token);
    Header header = jwt.getHeader();
    Object body = jwt.getBody();
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body);
}
```

```
header={typ=JWT, alg=HS256},body={sex=男, userId=10002, jti=jwt2, username=张三},signature=9TC0U77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98
{typ=JWT, alg=HS256}
{sex=男, userId=10002, jti=jwt2, username=张三}
```

```java
/**
 * 生成jwt令牌，基于RS256签名算法，错误
 */
@Test
void test7()
{
    Map<String, Object> head = new HashMap<>();
    head.put("alg", SignatureAlgorithm.RS256.getValue());
    head.put("typ", "JWT");

    Map<String, Object> body = new HashMap<>();
    body.put("userId", "10003");
    body.put("username", "张三");
    body.put("sex", "男");

    String token = Jwts.builder()
            .setHeader(head)
            .setClaims(body)
            .setId("jwt3")
            .signWith(SignatureAlgorithm.RS256, "123456")
            .compact();
    System.out.println(token);
}
```

```
java.lang.IllegalArgumentException: Base64-encoded key bytes may only be specified for HMAC signatures.  If using RSA or Elliptic Curve, use the signWith
 (SignatureAlgorithm, Key) method instead.

    at io.jsonwebtoken.lang.Assert.isTrue(Assert.java:38)
    at io.jsonwebtoken.impl.DefaultJwtBuilder.signWith(DefaultJwtBuilder.java:98)
    at mao.JwtTest.test7(JwtTest.java:175) <31 个内部行>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 个内部行>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <26 个内部行>
```

需要先生成秘钥/公钥 对

```java
/**
 * 生成自己的 秘钥/公钥 对
 *
 * @throws Exception 异常
 */
@Test
public void test8() throws Exception
{
    //自定义 随机密码,  请修改这里
    String password = "123456";

    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
    SecureRandom secureRandom = new SecureRandom(password.getBytes());
    keyPairGenerator.initialize(1024, secureRandom);
    KeyPair keyPair = keyPairGenerator.genKeyPair();

    byte[] publicKeyBytes = keyPair.getPublic().getEncoded();
    byte[] privateKeyBytes = keyPair.getPrivate().getEncoded();

    FileUtil.writeBytes(publicKeyBytes, "./pub.key");
    FileUtil.writeBytes(privateKeyBytes, "./pri.key");
}
```

```java
    //获取私钥
    public PrivateKey getPriKey() throws Exception
    {
//        InputStream inputStream =
//
this.getClass().getClassLoader().getResourceAsStream("pri.key");
        FileInputStream inputStream = new FileInputStream("./pri.key");
        DataInputStream dataInputStream = new DataInputStream(inputStream);
        byte[] keyBytes = new byte[inputStream.available()];
        dataInputStream.readFully(keyBytes);
```

```
        PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        return keyFactory.generatePrivate(pkcs8EncodedKeySpec);
    }

    //获取公钥
    public PublicKey getPubKey() throws Exception
    {
//        InputStream inputStream =
//this.getClass().getClassLoader().getResourceAsStream("pub.key");
        FileInputStream inputStream = new FileInputStream("./pub.key");
        DataInputStream dataInputStream = new DataInputStream(inputStream);
        byte[] keyBytes = new byte[inputStream.available()];
        dataInputStream.readFully(keyBytes);
        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        return keyFactory.generatePublic(spec);
    }

    /**
     * 生成jwt令牌，基于RS256签名算法
     */
    @Test
    void test9() throws Exception
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", SignatureAlgorithm.RS256.getValue());
        head.put("typ", "JWT");

        Map<String, Object> body = new HashMap<>();
        body.put("userId", "10003");
        body.put("username", "张三");
        body.put("sex", "男");

        String token = Jwts.builder()
                .setHeader(head)
                .setClaims(body)
                .setId("jwt3")
                .signWith(SignatureAlgorithm.RS256, getPriKey())
                .compact();
        System.out.println(token);

 //eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMyIsImp0aSI6Imp3dDMiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.Ke2oOWFNNQp71Sdd056bP2Z2CywxfaV4M9OUtsPNBmrLWSLNOkqUao3DiTdX2kLMMWjVQ4THnCQHRiJhXa2uPX6qLfNPHhCC1unYFBlU17WAPSfpp3BeEF4UK3G5GOiamLFghiowlwG84_3AuNFOj8JZXY4Beq_FpT9PSo1608M
    }
```

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMyIsImp0aSI6Imp3dDMiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.Ke2oOWFNNQp71Sdd056bP2Z2CywxfaV4M9OUtsPNBmrLWSLNOkqUao3DiTdX2kLMMWjVQ4THnCQHRiJhXa2uPX6qLfNPHhCC1unYFBlU17WAPSfpp3BeEF4UK3G5GOiamLFghiowlwG84_3AuNFOj8JZXY4Beq_FpT9PSo1608M

进程已结束，退出代码为 0
```

```java
/**
 * 解析jwt令牌，基于RS256签名算法
 */
@Test
void test10() throws Exception
{
    String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9" +

 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMyIsImp0aSI6Imp3dDMiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9" +

 ".Ke2oOWFNNQp71Sdd056bP2Z2CywxfaV4M9OUtsPNBmrLWSLNOkqUao3DiTdX2kLMMWjVQ4" +

 "THnCQHRiJhXa2uPX6qLfNPHhCC1unYFBlU17WAPSfpp3BeEF4UK3G5GOiamLFghiowlwG84_3AuNFOj8JZXY4Beq_FpT9PSo1608M";

    Jwt jwt = Jwts.parser()
            .setSigningKey(getPubKey())
            .parse(token);
    Header header = jwt.getHeader();
    Object body = jwt.getBody();
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body);
}
```

```
header={typ=JWT, alg=RS256},body={sex=男, userId=10003, jti=jwt3, username=张三},
 signature
 =Ke2oOWFNNQp71Sdd056bP2Z2CywxfaV4M9OUtsPNBmrLWSLNOkqUao3DiTdX2kLMMWjVQ4THnCQHRiJhXa2uPX6qLfNPHhCC1un
 eq_FpT9PSo1608M
{typ=JWT, alg=RS256}
{sex=男, userId=10003, jti=jwt3, username=张三}
```

```java
/**
 * 生成jwt令牌，基于RS256签名算法，带过期时间，解析过期的情况
 */
@Test
void test11() throws Exception
{
    Map<String, Object> head = new HashMap<>();
    head.put("alg", SignatureAlgorithm.RS256.getValue());
    head.put("typ", "JWT");

```

```
    Map<String, Object> body = new HashMap<>();
    body.put("userId", "10004");
    body.put("username", "张三");
    body.put("sex", "男");

    String token = Jwts.builder()
            .setHeader(head)
            .setClaims(body)
            .setExpiration(new Date(new Date().getTime() + 2 * 1000))//2秒
            .setId("jwt4")
            .signWith(SignatureAlgorithm.RS256, getPriKey())
            .compact();
    System.out.println(token);


    Thread.sleep(2000);

    Jwt jwt = Jwts.parser()
            .setSigningKey(getPubKey())
            .parse(token);
    Header header = jwt.getHeader();
    Object body2 = jwt.getBody();
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body2);
}
```


```
/**
 * 生成jwt令牌，基于RS256签名算法，带过期时间，解析没有过期的情况
 */
@Test
void test12() throws Exception
{
    Map<String, Object> head = new HashMap<>();
    head.put("alg", SignatureAlgorithm.RS256.getValue());
    head.put("typ", "JWT");

    Map<String, Object> body = new HashMap<>();
    body.put("userId", "10004");
    body.put("username", "张三");
```

```
    body.put("sex", "男");

    String token = Jwts.builder()
            .setHeader(head)
            .setClaims(body)
            .setExpiration(new Date(new Date().getTime() + 2 * 1000))//2秒
            .setId("jwt4")
            .signWith(SignatureAlgorithm.RS256, getPriKey())
            .compact();
    System.out.println(token);


    //Thread.sleep(2000);

    System.out.println("\n-------\n");

    Jwt jwt = Jwts.parser()
            .setSigningKey(getPubKey())
            .parse(token);
    Header header = jwt.getHeader();
    Object body2 = jwt.getBody();
    System.out.println(jwt);
    System.out.println(header);
    System.out.println(body2);
}
```

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzZXgiOiLnlLciLCJleHAiOjE2NjczOTQxMjIsInVzZXJJZCI6IjEwMDA0IiwianRpIjoiand0NCIsInVzZXJuYW1lIjoi5byg5LiJIn0
 .LluCrQmRuolghlhcsoPgKcPoPbg4dutyUywx-5iRx-quBd-TsjfpB9D7-6W171dEc1y5abxZ5XqVlBzyANI4lwJNlDWFDaMPIzsWtZqq7gQNGJMPX3EY4WL_gTXQuVyCO7JgdTU
 -685EAK6IwsFwNwHSl4FvCWprKmvhFhDzMvI

-------

header={typ=JWT, alg=RS256},body={sex=男, exp=1667394122, userId=10004, jti=jwt4, username=张三},
 signature=LluCrQmRuolghlhcsoPgKcPoPbg4dutyUywx-5iRx-quBd-TsjfpB9D7-6W171dEc1y5abxZ5XqVlBzyANI4lwJNlDWFDaMPIzsWtZqq7gQNGJMPX3EY4WL_gTXQuVyCO7JgdTU
 -685EAK6IwsFwNwHSl4FvCWprKmvhFhDzMvI
{typ=JWT, alg=RS256}
{sex=男, exp=1667394122, userId=10004, jti=jwt4, username=张三}
```

全部源码：

```
package mao;

import cn.hutool.core.io.FileUtil;
import io.jsonwebtoken.Header;
import io.jsonwebtoken.Jwt;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.junit.jupiter.api.Test;

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.InputStream;
```

```
import java.security.*;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

/**
 * Project name(项目名称)：jwt_demo
 * Package(包名): mao
 * Class(类名): JwtTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 13:40
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class JwtTest
{

    /**
     * 生成token，不使用签名
     */
    @Test
    void test1()
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", "none");
        head.put("typ", "JWT");

        Map<String, Object> body = new HashMap<>();
        body.put("userId", "10001");
        body.put("username", "张三");
        body.put("sex", "男");

        String token = Jwts.builder()
                .setHeader(head)
                .setClaims(body)
                .setId("jwt1")
                .compact();
        System.out.println(token);

 //eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMSIsImp0aSI6Imp3dDEiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.
    }

    /**
     * 解析token，不使用签名
     */
    @Test
    void test2()
    {
        Jwt jwt =
Jwts.parser().parse("eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0." +
```

```
 "eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMSIsImp0aSI6Imp3dDEiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.");
        Header header = jwt.getHeader();
        Object body = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
        System.out.println(body);
    }


    /**
     * 生成token，使用hs256签名算法
     */
    @Test
    void test3()
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", SignatureAlgorithm.HS256.getValue());
        head.put("typ", "JWT");

        Map<String, Object> body = new HashMap<>();
        body.put("userId", "10002");
        body.put("username", "张三");
        body.put("sex", "男");

        String token = Jwts.builder()
                .setHeader(head)
                .setClaims(body)
                .setId("jwt2")
                .signWith(SignatureAlgorithm.HS256, "123456")
                .compact();
        System.out.println(token);
        //eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
        //
.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9
        // .9TCOU77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98
    }

    /**
     * 解析token，使用hs256签名算法，不设置SigningKey的情况
     */
    @Test
    void test4()
    {
        String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9" +
 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9." +
                "9TCOU77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98";

        Jwt jwt = Jwts.parser()
                .parse(token);
        Header header = jwt.getHeader();
        Object body = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
```

```
        System.out.println(body);
    }


    /**
     * 解析token，使用hs256签名算法，SigningKey错误的情况
     */
    @Test
    void test5()
    {
        String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9" +
 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9." +
                "9TCOU77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98";

        Jwt jwt = Jwts.parser()
                .setSigningKey("1236")
                .parse(token);
        Header header = jwt.getHeader();
        Object body = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
        System.out.println(body);
    }

    /**
     * 解析token，使用hs256签名算法，SigningKey正确的情况
     */
    @Test
    void test6()
    {
        String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9" +
 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMiIsImp0aSI6Imp3dDIiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9." +
                "9TCOU77uYueqnUdU_we2yVUZ6uj9mrsLPhjr4gB2v98";

        Jwt jwt = Jwts.parser()
                .setSigningKey("123456")
                .parse(token);
        Header header = jwt.getHeader();
        Object body = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
        System.out.println(body);
    }


    /**
     * 生成jwt令牌，基于RS256签名算法，错误
     */
    @Test
    void test7()
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", SignatureAlgorithm.RS256.getValue());
        head.put("typ", "JWT");
```

```
173
174         Map<String, Object> body = new HashMap<>();
175         body.put("userId", "10003");
176         body.put("username", "张三");
177         body.put("sex", "男");
178
179         String token = Jwts.builder()
180                 .setHeader(head)
181                 .setClaims(body)
182                 .setId("jwt3")
183                 .signWith(SignatureAlgorithm.RS256, "123456")
184                 .compact();
185         System.out.println(token);
186     }
187
188
189     /**
190      * 生成自己的 秘钥/公钥 对
191      *
192      * @throws Exception 异常
193      */
194     @Test
195     public void test8() throws Exception
196     {
197         //自定义 随机密码,  请修改这里
198         String password = "123456";
199
200         KeyPairGenerator keyPairGenerator =
    KeyPairGenerator.getInstance("RSA");
201         SecureRandom secureRandom = new SecureRandom(password.getBytes());
202         keyPairGenerator.initialize(1024, secureRandom);
203         KeyPair keyPair = keyPairGenerator.genKeyPair();
204
205         byte[] publicKeyBytes = keyPair.getPublic().getEncoded();
206         byte[] privateKeyBytes = keyPair.getPrivate().getEncoded();
207
208         FileUtil.writeBytes(publicKeyBytes, "./pub.key");
209         FileUtil.writeBytes(privateKeyBytes, "./pri.key");
210     }
211
212     //获取私钥
213     public PrivateKey getPriKey() throws Exception
214     {
215 //         InputStream inputStream =
216 //
    this.getClass().getClassLoader().getResourceAsStream("pri.key");
217         FileInputStream inputStream = new FileInputStream("./pri.key");
218         DataInputStream dataInputStream = new DataInputStream(inputStream);
219         byte[] keyBytes = new byte[inputStream.available()];
220         dataInputStream.readFully(keyBytes);
221         PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new
    PKCS8EncodedKeySpec(keyBytes);
222         KeyFactory keyFactory = KeyFactory.getInstance("RSA");
223         return keyFactory.generatePrivate(pkcs8EncodedKeySpec);
224     }
225
226     //获取公钥
227     public PublicKey getPubKey() throws Exception
```

```
    {
//        InputStream inputStream =
//this.getClass().getClassLoader().getResourceAsStream("pub.key");
        FileInputStream inputStream = new FileInputStream("./pub.key");
        DataInputStream dataInputStream = new DataInputStream(inputStream);
        byte[] keyBytes = new byte[inputStream.available()];
        dataInputStream.readFully(keyBytes);
        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        return keyFactory.generatePublic(spec);
    }

    /**
     * 生成jwt令牌，基于RS256签名算法
     */
    @Test
    void test9() throws Exception
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", SignatureAlgorithm.RS256.getValue());
        head.put("typ", "JWT");

        Map<String, Object> body = new HashMap<>();
        body.put("userId", "10003");
        body.put("username", "张三");
        body.put("sex", "男");

        String token = Jwts.builder()
                .setHeader(head)
                .setClaims(body)
                .setId("jwt3")
                .signWith(SignatureAlgorithm.RS256, getPriKey())
                .compact();
        System.out.println(token);

 //eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwM
        //yIsImp0aSI6Imp3dDMiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9.Ke2oOWFNNQp71Sdd056bP2Z2
        //CywxfaV4M9OUtsPNBmrLWSLNOkqUao3DiTdX2kLMMWjVQ4THnCQHRiJhXa2uPX6qLfNPHh
        //CC1unYFBlU17WAPSfpp3BeEF4UK3G5GOiamLFghiowlwG84_3AuNFOj8JZXY4Beq_FpT9PSo1608M
    }

    /**
     * 解析jwt令牌，基于RS256签名算法
     */
    @Test
    void test10() throws Exception
    {
        String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9" +

 ".eyJzZXgiOiLnlLciLCJ1c2VySWQiOiIxMDAwMyIsImp0aSI6Imp3dDMiLCJ1c2VybmFtZSI6IuW8oOS4iSJ9" +
```

```
 ".Ke2oOWFNNQp71Sdd056bP2Z2CywxfaV4M9OUtsPNBmrLWSLNOkqUao3DiTdX2kLMMWjVQ4"
+
 "THnCQHRiJhXa2uPX6qLfNPHhCC1unYFBlU17WAPSfpp3BeEF4UK3G5GOiamLFghiowlwG84_3
AuNFOj8JZXY4Beq_FpT9PSo16O8M";

        Jwt jwt = Jwts.parser()
                .setSigningKey(getPubKey())
                .parse(token);
        Header header = jwt.getHeader();
        Object body = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
        System.out.println(body);
    }

    /**
     * 生成jwt令牌，基于RS256签名算法，带过期时间，解析过期的情况
     */
    @Test
    void test11() throws Exception
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", SignatureAlgorithm.RS256.getValue());
        head.put("typ", "JWT");

        Map<String, Object> body = new HashMap<>();
        body.put("userId", "10004");
        body.put("username", "张三");
        body.put("sex", "男");

        String token = Jwts.builder()
                .setHeader(head)
                .setClaims(body)
                .setExpiration(new Date(new Date().getTime() + 2 *
1000))//2秒
                .setId("jwt4")
                .signWith(SignatureAlgorithm.RS256, getPriKey())
                .compact();
        System.out.println(token);


        Thread.sleep(2000);

        Jwt jwt = Jwts.parser()
                .setSigningKey(getPubKey())
                .parse(token);
        Header header = jwt.getHeader();
        Object body2 = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
        System.out.println(body2);
    }


    /**
     * 生成jwt令牌，基于RS256签名算法，带过期时间，解析没有过期的情况
```

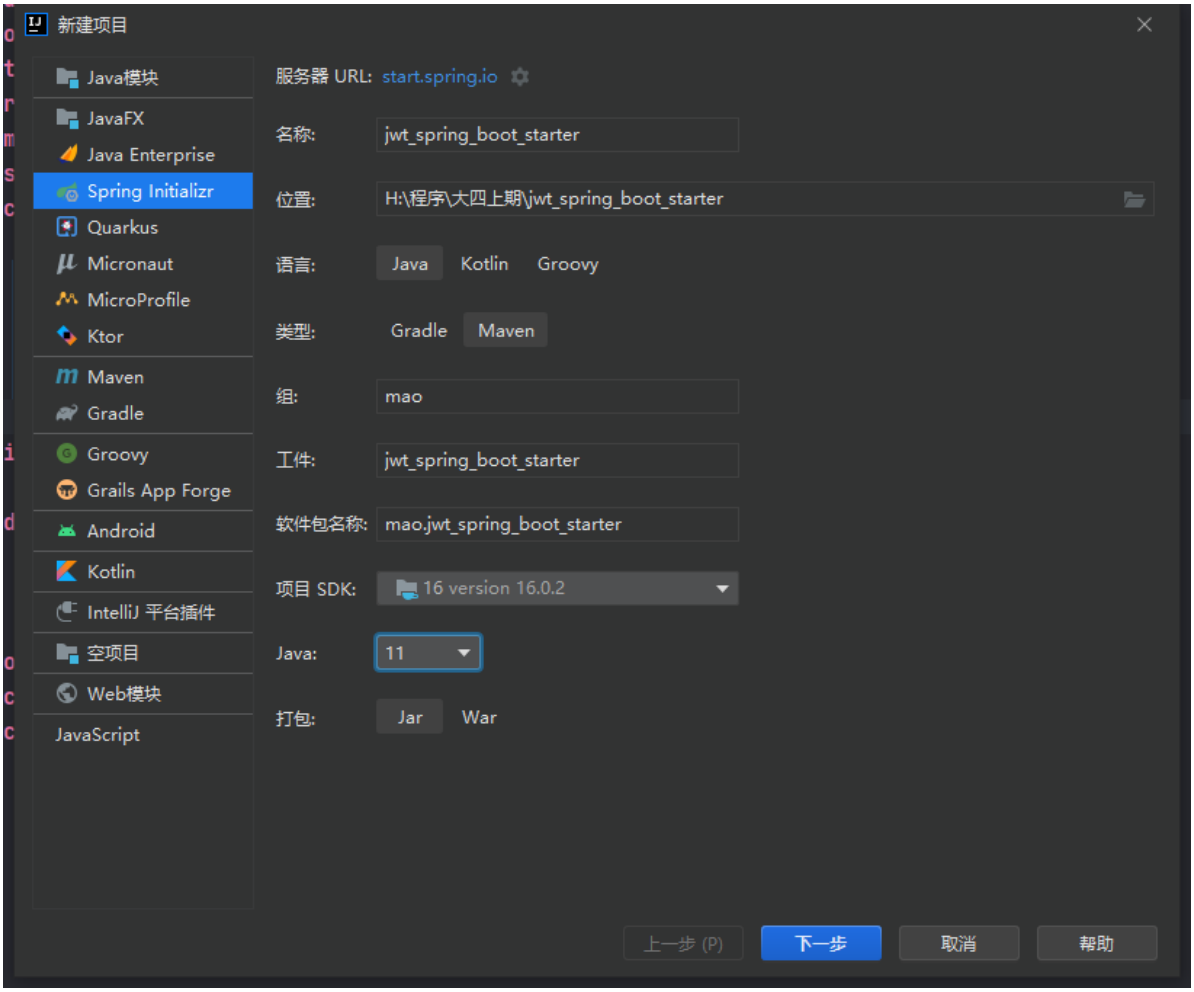
```
     */
    @Test
    void test12() throws Exception
    {
        Map<String, Object> head = new HashMap<>();
        head.put("alg", SignatureAlgorithm.RS256.getValue());
        head.put("typ", "JWT");

        Map<String, Object> body = new HashMap<>();
        body.put("userId", "10004");
        body.put("username", "张三");
        body.put("sex", "男");

        String token = Jwts.builder()
                .setHeader(head)
                .setClaims(body)
                .setExpiration(new Date(new Date().getTime() + 2 *
1000))//2秒
                .setId("jwt4")
                .signWith(SignatureAlgorithm.RS256, getPriKey())
                .compact();
        System.out.println(token);


        //Thread.sleep(2000);

        System.out.println("\n-------\n");

        Jwt jwt = Jwts.parser()
                .setSigningKey(getPubKey())
                .parse(token);
        Header header = jwt.getHeader();
        Object body2 = jwt.getBody();
        System.out.println(jwt);
        System.out.println(header);
        System.out.println(body2);
    }
}
```

自定义spring boot starter

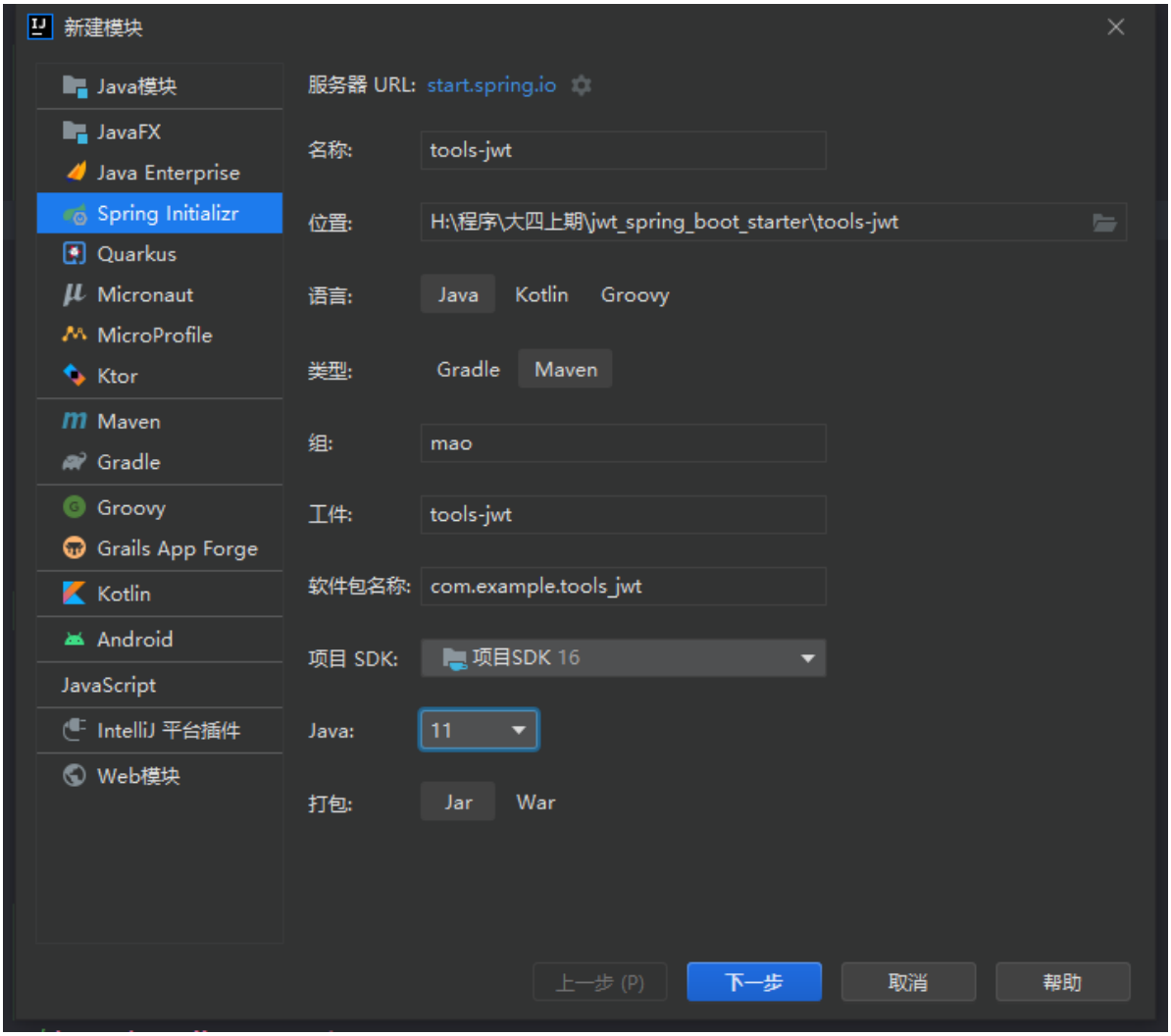
开发starter

第一步：初始化项目

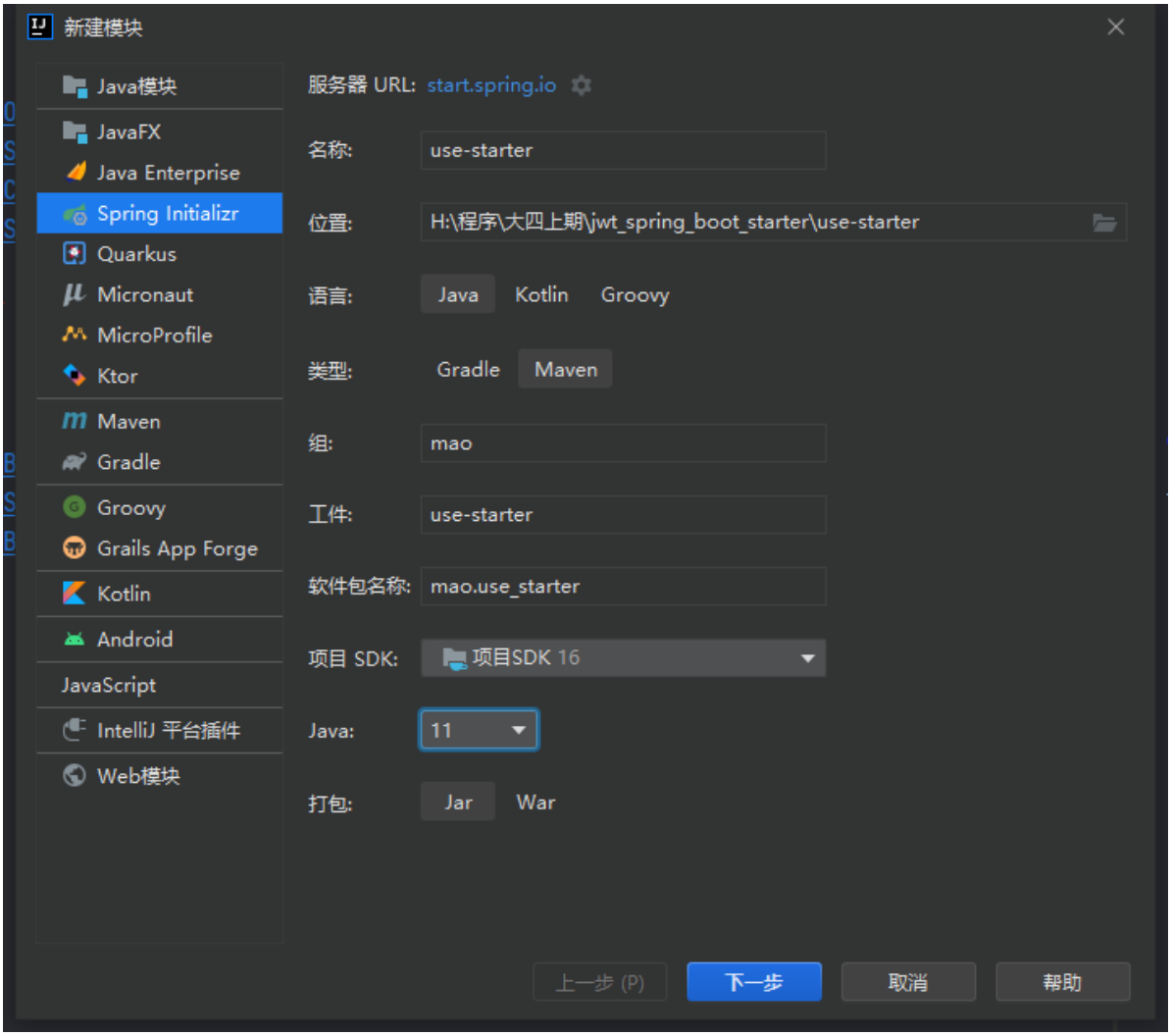
创建父工程jwt_spring_boot_starter



创建子工程tools-jwt



创建子工程use-starter



第二步：修改pom文件

父工程jwt_spring_boot_starter的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>mao</groupId>
    <artifactId>jwt_spring_boot_starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```
    <name>jwt_spring_boot_starter</name>
    <description>jwt_spring_boot_starter</description>
    <packaging>pom</packaging>

    <properties>
        <java.version>11</java.version>
    </properties>

    <modules>
        <module>tools-jwt</module>
        <module>use-starter</module>
    </modules>

    <dependencies>

    </dependencies>

    <dependencyManagement>
        <dependencies>

        </dependencies>
    </dependencyManagement>

</project>
```

子工程tools-jwt的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <artifactId>jwt_spring_boot_starter</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>tools-jwt</artifactId>
    <name>tools-jwt</name>
    <description>tools-jwt</description>
    <properties>

    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

```

```
        <!--jwt 依赖-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.8.0</version>
        </dependency>

        <!--java 8 版本不需要添加-->
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-impl</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-core</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>javax.activation</groupId>
            <artifactId>activation</artifactId>
            <version>1.1.1</version>
        </dependency>

        <dependency>
            <groupId>com.github.xiaoymin</groupId>
            <artifactId>knife4j-spring-boot-starter</artifactId>
            <version>2.0.1</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <skip>true</skip>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
                <configuration>
                    <encoding>UTF-8</encoding>
                    <!-- 过滤后缀为pem、pfx的证书文件 -->
```

```
                    <nonFilteredFileExtensions>
 <nonFilteredFileExtension>pem</nonFilteredFileExtension>
 <nonFilteredFileExtension>pfx</nonFilteredFileExtension>
 <nonFilteredFileExtension>p12</nonFilteredFileExtension>
 <nonFilteredFileExtension>key</nonFilteredFileExtension>
                    </nonFilteredFileExtensions>
                </configuration>
            </plugin>
        </plugins>
    </build>


</project>

```

子工程use-starter的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <artifactId>jwt_spring_boot_starter</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>use-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>use-starter</name>
    <description>use-starter</description>
    <properties>

    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

```
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第三步：编写实体类JwtUserInfo

```
package com.example.tools_jwt.entity;

import java.io.Serializable;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.entity
 * Class(类名): JwtUserInfo
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:12
 * Version(版本): 1.0
 * Description(描述)： 无
 */


public class JwtUserInfo implements Serializable
{
    /**
     * 账号id
     */
    private Long userId;
    /**
     * 账号
     */
    private String account;
    /**
     * 姓名
     */
    private String name;

    /**
```

```
     * 当前登录人单位id
     */
    private Long orgId;

    /**
     * 当前登录人岗位ID
     */
    private Long stationId;

    /**
     * Instantiates a new Jwt user info.
     */
    public JwtUserInfo()
    {

    }

    /**
     * Instantiates a new Jwt user info.
     *
     * @param userId    the user id
     * @param account   the account
     * @param name      the name
     * @param orgId     the org id
     * @param stationId the station id
     */
    public JwtUserInfo(Long userId, String account, String name, Long
orgId, Long stationId)
    {
        this.userId = userId;
        this.account = account;
        this.name = name;
        this.orgId = orgId;
        this.stationId = stationId;
    }

    /**
     * Gets user id.
     *
     * @return the user id
     */
    public Long getUserId()
    {
        return userId;
    }

    /**
     * Sets user id.
     *
     * @param userId the user id
     */
    public void setUserId(Long userId)
    {
        this.userId = userId;
    }

    /**
     * Gets account.
```

```
92          *
93          * @return the account
94          */
95         public String getAccount()
96         {
97             return account;
98         }
99
100        /**
101         * Sets account.
102         *
103         * @param account the account
104         */
105        public void setAccount(String account)
106        {
107            this.account = account;
108        }
109
110        /**
111         * Gets name.
112         *
113         * @return the name
114         */
115        public String getName()
116        {
117            return name;
118        }
119
120        /**
121         * Sets name.
122         *
123         * @param name the name
124         */
125        public void setName(String name)
126        {
127            this.name = name;
128        }
129
130        /**
131         * Gets org id.
132         *
133         * @return the org id
134         */
135        public Long getOrgId()
136        {
137            return orgId;
138        }
139
140        /**
141         * Sets org id.
142         *
143         * @param orgId the org id
144         */
145        public void setOrgId(Long orgId)
146        {
147            this.orgId = orgId;
148        }
149
```

```
    /**
     * Gets station id.
     *
     * @return the station id
     */
    public Long getStationId()
    {
        return stationId;
    }

    /**
     * Sets station id.
     *
     * @param stationId the station id
     */
    public void setStationId(Long stationId)
    {
        this.stationId = stationId;
    }

    @Override
    public String toString()
    {
        final StringBuffer sb = new StringBuffer("JwtUserInfo{");
        sb.append("userId=").append(userId);
        sb.append(", account='").append(account).append('\'');
        sb.append(", name='").append(name).append('\'');
        sb.append(", orgId=").append(orgId);
        sb.append(", stationId=").append(stationId);
        sb.append('}');
        return sb.toString();
    }
}
```

第四步：编写实体类Token

```
package com.example.tools_jwt.entity;

import io.swagger.annotations.ApiModelProperty;

import java.io.Serializable;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.entity
 * Class(类名): Token
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:15
```

```
 * Version(版本): 1.0
 * Description(描述)： 无
 */
public class Token implements Serializable
{
    private static final long serialVersionUID = -8482946147572784305L;

    /**
     * token
     */
    @ApiModelProperty(value = "token")
    private String token;
    /**
     * 有效时间：单位：秒
     */
    @ApiModelProperty(value = "token的有效期")
    private Integer expire;

    /**
     * Instantiates a new Token.
     */
    public Token()
    {

    }

    /**
     * Instantiates a new Token.
     *
     * @param token  the token
     * @param expire the expire
     */
    public Token(String token, Integer expire)
    {
        this.token = token;
        this.expire = expire;
    }

    /**
     * Gets token.
     *
     * @return the token
     */
    public String getToken()
    {
        return token;
    }

    /**
     * Sets token.
     *
     * @param token the token
     */
    public void setToken(String token)
    {
        this.token = token;
    }

```

```
    /**
     * Gets expire.
     *
     * @return the expire
     */
    public Integer getExpire()
    {
        return expire;
    }

    /**
     * Sets expire.
     *
     * @param expire the expire
     */
    public void setExpire(Integer expire)
    {
        this.expire = expire;
    }
}
```

第五步：编写工具类RsaKeyHelper

```
package com.example.tools_jwt.utils;

import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.utils
 * Class(类名): RsaKeyHelper
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:19
 * Version(版本): 1.0
 * Description(描述)： Rsa key 帮助类
 */

public class RsaKeyHelper
{
```

```
29
30      /**
31       * 获取公钥,用于解析token
32       *
33       * @param filename 文件名
34       * @return {@link PublicKey}
35       * @throws IOException              ioexception
36       * @throws NoSuchAlgorithmException 没有这样算法异常
37       * @throws InvalidKeySpecException  无效关键规范异常
38       */
39      public PublicKey getPublicKey(String filename) throws IOException,
    NoSuchAlgorithmException, InvalidKeySpecException
40      {
41          InputStream inputStream =
    this.getClass().getClassLoader().getResourceAsStream(filename);
42          if (inputStream == null)
43          {
44              throw new IOException("获取公钥时获取失败，可能是公钥文件不存在。当前路
    径: " + filename);
45          }
46          try (DataInputStream dataInputStream = new
    DataInputStream(inputStream))
47          {
48              byte[] keyBytes = new byte[inputStream.available()];
49              dataInputStream.readFully(keyBytes);
50              X509EncodedKeySpec x509EncodedKeySpec = new
    X509EncodedKeySpec(keyBytes);
51              KeyFactory keyFactory = KeyFactory.getInstance("RSA");
52              return keyFactory.generatePublic(x509EncodedKeySpec);
53          }
54      }
55
56
57      /**
58       * 获取私钥 用于生成token
59       *
60       * @param filename 文件名
61       * @return {@link PrivateKey}
62       * @throws IOException              ioexception
63       * @throws NoSuchAlgorithmException 没有这样算法异常
64       * @throws InvalidKeySpecException  无效关键规范异常
65       */
66      public PrivateKey getPrivateKey(String filename)
67              throws IOException, NoSuchAlgorithmException,
    InvalidKeySpecException
68      {
69          InputStream inputStream =
    this.getClass().getClassLoader().getResourceAsStream(filename);
70          if (inputStream == null)
71          {
72              throw new IOException("获取私钥时获取失败，可能是私钥文件不存在。当前路
    径: " + filename);
73          }
74          try (DataInputStream dataInputStream = new
    DataInputStream(inputStream))
75          {
76              byte[] keyBytes = new byte[inputStream.available()];
77              dataInputStream.readFully(keyBytes);
```

```
78
79              PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new
PKCS8EncodedKeySpec(keyBytes);
80              KeyFactory keyFactory = KeyFactory.getInstance("RSA");
81              return keyFactory.generatePrivate(pkcs8EncodedKeySpec);
82          }
83      }
84  }
```

第六步：编写工具类NumberHelper

```
1   package com.example.tools_jwt.utils;
2
3   import java.util.function.Function;
4
5   /**
6    * Project name(项目名称)：jwt_spring_boot_starter
7    * Package(包名): com.example.tools_jwt.utils
8    * Class(类名): NumberHelper
9    * Author(作者）: mao
10   * Author QQ：1296193245
11   * GitHub：https://github.com/maomao124/
12   * Date(创建日期)： 2022/11/2
13   * Time(创建时间)： 22:27
14   * Version(版本): 1.0
15   * Description(描述)： 无
16   */
17
18  public class NumberHelper
19  {
20      private static <T, R> R valueOfDef(T t, Function<T, R> function, R def)
21      {
22          try
23          {
24              return function.apply(t);
25          }
26          catch (Exception e)
27          {
28              return def;
29          }
30      }
31
32      public static Long longValueOfNil(String value)
33      {
34          return valueOfDef(value, (val) -> Long.valueOf(val), null);
35      }
36
37      public static Long longValueOf0(String value)
38      {
39          return valueOfDef(value, (val) -> Long.valueOf(val), 0L);
40      }
```

```
41
42      public static Long longValueOfNil(Object value)
43      {
44          return valueOfDef(value, (val) -> Long.valueOf(val.toString()),
    null);
45      }
46
47      public static Long longValueOf0(Object value)
48      {
49          return valueOfDef(value, (val) -> Long.valueOf(val.toString()), 0L);
50      }
51
52      public static Boolean boolValueOf0(Object value)
53      {
54          return valueOfDef(value, (val) -> Boolean.valueOf(val.toString()),
    false);
55      }
56
57      public static Integer intValueOfNil(String value)
58      {
59          return valueOfDef(value, (val) -> Integer.valueOf(val), null);
60      }
61
62      public static Integer intValueOf0(String value)
63      {
64          return intValueOf(value, 0);
65      }
66
67      public static Integer intValueOf(String value, Integer def)
68      {
69          return valueOfDef(value, (val) -> Integer.valueOf(val), def);
70      }
71
72      public static Integer intValueOfNil(Object value)
73      {
74          return valueOfDef(value, (val) -> Integer.valueOf(val.toString()),
    null);
75      }
76
77      public static Integer intValueOf0(Object value)
78      {
79          return valueOfDef(value, (val) -> Integer.valueOf(val.toString()),
    0);
80      }
81
82      public static Integer getOrDef(Integer val, Integer def)
83      {
84          return val == null ? def : val;
85      }
86
87      public static Long getOrDef(Long val, Long def)
88      {
89          return val == null ? def : val;
90      }
91
92      public static Boolean getOrDef(Boolean val, Boolean def)
93      {
94          return val == null ? def : val;
```

```
    }
}
```

第七步：编写工具类StrHelper

```
package com.example.tools_jwt.utils;

import cn.hutool.core.util.StrUtil;

import java.net.URLDecoder;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.utils
 * Class(类名): StrHelper
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:28
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class StrHelper
{
    public static String getObjectValue(Object obj)
    {
        return obj == null ? "" : obj.toString();
    }

    public static String encode(String value)
    {
        try
        {
            return URLEncoder.encode(value, StandardCharsets.UTF_8);
        }
        catch (Exception e)
        {
            return "";
        }
    }

    public static String decode(String value)
    {
        try
        {
            return URLDecoder.decode(value, StandardCharsets.UTF_8);
        }
```

```
        catch (Exception e)
        {
            return "";
        }
    }

    public static String getOrDef(String val, String def)
    {
        return StrUtil.isEmpty(val) ? def : val;
    }
}
```

第八步：编写接口BaseException

```
package com.example.tools_jwt.exception;

/**
 * 异常接口类
 */
public interface BaseException
{

    /**
     * 统一参数验证异常码
     */
    int BASE_VALID_PARAM = -9;

    /**
     * 返回异常信息
     *
     * @return {@link String}
     */
    String getMessage();

    /**
     * 返回异常编码
     *
     * @return int
     */
    int getCode();

}
```

第九步：编写接口BaseExceptionCode

```java
package com.example.tools_jwt.exception;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.exception
 * Interface(接口名): BaseExceptionCode
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:32
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public interface BaseExceptionCode
{
    /**
     * 异常编码
     *
     * @return int
     */
    int getCode();

    /**
     * 异常消息
     *
     * @return String
     */
    String getMsg();
}
```

第十步：编写类BaseUncheckedException

```java
package com.example.tools_jwt.exception;

/**
 * 非运行期异常基类，所有自定义非运行时异常继承该类
 */
public class BaseUncheckedException extends RuntimeException implements
BaseException
{

    private static final long serialVersionUID = -778887391066124051L;

    /**
     * 异常信息
```

```
     */
    protected String message;

    /**
     * 具体异常码
     */
    protected int code;

    public BaseUncheckedException(int code, String message)
    {
        super(message);
        this.code = code;
        this.message = message;
    }

    public BaseUncheckedException(int code, String format, Object... args)
    {
        super(String.format(format, args));
        this.code = code;
        this.message = String.format(format, args);
    }


    @Override
    public String getMessage()
    {
        return message;
    }

    @Override
    public int getCode()
    {
        return code;
    }
}
```

第十一步：编写类ExceptionCode

```
package com.example.tools_jwt.exception;

/**
 * 全局错误码 10000-15000
 * <p>
 * 预警异常编码     范围：  30000~34999
 * 标准服务异常编码 范围：35000~39999
 * 邮件服务异常编码 范围：40000~44999
 * 短信服务异常编码 范围：45000~49999
 * 权限服务异常编码 范围：50000-59999
 * 文件服务异常编码 范围：60000~64999
 * 日志服务异常编码 范围：65000~69999
 * 消息服务异常编码 范围：70000~74999
```

```
 * 开发者平台异常编码 范围：75000~79999
 * 搜索服务异常编码 范围：80000-84999
 * 共享交换异常编码 范围：85000-89999
 * 移动终端平台 异常码 范围：90000-94999
 * <p>
 * 安全保障平台     范围：          95000-99999
 * 软硬件平台 异常编码 范围：     100000-104999
 * 运维服务平台 异常编码 范围：   105000-109999
 * 统一监管平台异常 编码 范围：   110000-114999
 * 认证方面的异常编码   范围：115000-115999
 */

public enum ExceptionCode implements BaseExceptionCode
{

    //系统相关 start
    SUCCESS(0, "成功"),
    SYSTEM_BUSY(-1, "系统繁忙~请稍后再试~"),
    SYSTEM_TIMEOUT(-2, "系统维护中~请稍后再试~"),
    PARAM_EX(-3, "参数类型解析异常"),
    SQL_EX(-4, "运行SQL出现异常"),
    NULL_POINT_EX(-5, "空指针异常"),
    ILLEGALA_ARGUMENT_EX(-6, "无效参数异常"),
    MEDIA_TYPE_EX(-7, "请求类型异常"),
    LOAD_RESOURCES_ERROR(-8, "加载资源出错"),
    BASE_VALID_PARAM(-9, "统一验证参数异常"),
    OPERATION_EX(-10, "操作异常"),


    OK(200, "OK"),
    BAD_REQUEST(400, "错误的请求"),
    /**
     * {@code 401 Unauthorized}.
     *
     * @see <a href="http://tools.ietf.org/html/rfc7235#section-
3.1">HTTP/1.1: Authentication, section 3.1</a>
     */
    UNAUTHORIZED(401, "未经授权"),
    /**
     * {@code 404 Not Found}.
     *
     * @see <a href="http://tools.ietf.org/html/rfc7231#section-
6.5.4">HTTP/1.1: Semantics and Content, section 6.5.4</a>
     */
    NOT_FOUND(404, "没有找到资源"),
    METHOD_NOT_ALLOWED(405, "不支持当前请求类型"),

    TOO_MANY_REQUESTS(429, "请求超过次数限制"),
    INTERNAL_SERVER_ERROR(500, "内部服务错误"),
    BAD_GATEWAY(502, "网关错误"),
    GATEWAY_TIMEOUT(504, "网关超时"),
    //系统相关 end

    REQUIRED_FILE_PARAM_EX(1001, "请求中必须至少包含一个有效文件"),
    //jwt token 相关 start

    JWT_TOKEN_EXPIRED(40001, "会话超时，请重新登录"),
    JWT_SIGNATURE(40002, "不合法的token，请认真比对 token 的签名"),
```

```
    JWT_ILLEGAL_ARGUMENT(40003, "缺少token参数"),
    JWT_GEN_TOKEN_FAIL(40004, "生成token失败"),
    JWT_PARSER_TOKEN_FAIL(40005, "解析token失败"),
    JWT_USER_INVALID(40006, "用户名或密码错误"),
    JWT_USER_ENABLED(40007, "用户已经被禁用！"),
    //jwt token 相关 end

    ;

    private int code;
    private String msg;

    ExceptionCode(int code, String msg)
    {
        this.code = code;
        this.msg = msg;
    }

    @Override
    public int getCode()
    {
        return code;
    }

    @Override
    public String getMsg()
    {
        return msg;
    }


    public ExceptionCode build(String msg, Object... param)
    {
        this.msg = String.format(msg, param);
        return this;
    }

    public ExceptionCode param(Object... param)
    {
        msg = String.format(msg, param);
        return this;
    }
}
```

第十二步：编写类BizException

```
package com.example.tools_jwt.exception;

/**
 * 业务异常
 * 用于在处理业务逻辑时，进行抛出的异常。
```

```java
 */
public class BizException extends BaseUncheckedException
{

    private static final long serialVersionUID = -3843907364558373817L;

    public BizException(String message)
    {
        super(-1, message);
    }

    public BizException(int code, String message)
    {
        super(code, message);
    }

    public BizException(int code, String message, Object... args)
    {
        super(code, message, args);
    }

    /**
     * 实例化异常
     *
     * @param code    自定义异常编码
     * @param message 自定义异常消息
     * @param args    已定义异常参数
     * @return
     */
    public static BizException wrap(int code, String message, Object...
args)
    {
        return new BizException(code, message, args);
    }

    public static BizException wrap(String message, Object... args)
    {
        return new BizException(-1, message, args);
    }

    public static BizException validFail(String message, Object... args)
    {
        return new BizException(-9, message, args);
    }

    public static BizException wrap(BaseExceptionCode ex)
    {
        return new BizException(ex.getCode(), ex.getMsg());
    }

    @Override
    public String toString()
    {
        return "BizException [message=" + message + ", code=" + code + "]";
    }

}
```

第十三步：编写常量工具类BaseContextConstants

```java
package com.itheima.pinda.context;

/**
 * 常量工具类
 *
 */
public class BaseContextConstants {
    /**
     *
     */
    public static final String TOKEN_NAME = "token";
    /**
     *
     */
    public static final String JWT_KEY_USER_ID = "userid";
    /**
     *
     */
    public static final String JWT_KEY_NAME = "name";
    /**
     *
     */
    public static final String JWT_KEY_ACCOUNT = "account";

    /**
     * 组织id
     */
    public static final String JWT_KEY_ORG_ID = "orgid";
    /**
     * 岗位id
     */
    public static final String JWT_KEY_STATION_ID = "stationid";

    /**
     * 动态数据库名前缀。   每个项目配置死的
     */
    public static final String DATABASE_NAME = "database_name";
}
```

第十四步：编写日期工具类DateUtils

```java
package com.example.tools_jwt.utils;

import com.example.tools_jwt.exception.BizException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.stream.Stream;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.utils
 * Class(类名): DateUtils
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:30
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class DateUtils
{
    /**
     * 日志
     */
    private static final Logger log =
LoggerFactory.getLogger(DateUtils.class);

    /**
     * 默认年格式
     */
    public final static String DEFAULT_YEAR_FORMAT = "yyyy";
    /**
     * 默认月格式
     */
    public final static String DEFAULT_MONTH_FORMAT = "yyyy-MM";
    /**
     * 默认月格式削减
     */
    public final static String DEFAULT_MONTH_FORMAT_SLASH = "yyyy/MM";
    /**
     * 默认月格式在
     */
    public final static String DEFAULT_MONTH_FORMAT_EN = "yyyy年MM月";
```

```
    /**
     * 默认星期格式
     */
    public final static String DEFAULT_WEEK_FORMAT = "yyyy-ww";
    /**
     * 默认星期格式在
     */
    public final static String DEFAULT_WEEK_FORMAT_EN = "yyyy年ww周";
    /**
     * 默认日期格式
     */
    public final static String DEFAULT_DATE_FORMAT = "yyyy-MM-dd";
    /**
     * 默认日期格式在
     */
    public final static String DEFAULT_DATE_FORMAT_EN = "yyyy年MM月dd日";
    /**
     * 默认日期时间格式
     */
    public final static String DEFAULT_DATE_TIME_FORMAT = "yyyy-MM-dd
HH:mm:ss";
    /**
     * 默认时间格式
     */
    public final static String DEFAULT_TIME_FORMAT = "HH:mm:ss";
    /**
     * 一天
     */
    public final static String DAY = "DAY";
    /**
     * 月
     */
    public final static String MONTH = "MONTH";
    /**
     * 周
     */
    public final static String WEEK = "WEEK";

    /**
     * 最大值月一天
     */
    public final static long MAX_MONTH_DAY = 30;
    /**
     * 最大值3.月一天
     */
    public final static long MAX_3_MONTH_DAY = 90;
    /**
     * 最大值一年一天
     */
    public final static long MAX_YEAR_DAY = 365;


    private DateUtils()
    {
    }
//--格式化日期start----------------------------------------

    /**
```

```
     * 格式化日期,返回格式为 yyyy-MM
     *
     * @param date 日期
     * @return
     */
    public static String format(LocalDateTime date, String pattern)
    {
        if (date == null)
        {
            date = LocalDateTime.now();
        }
        if (pattern == null)
        {
            pattern = DEFAULT_MONTH_FORMAT;
        }
        return date.format(DateTimeFormatter.ofPattern(pattern));
    }

    /**
     * 根据传入的格式格式化日期.默认格式为MM月dd日
     *
     * @param d 日期
     * @param f 格式
     * @return
     */
    public static String format(Date d, String f)
    {
        Date date = d;
        String format = f;
        if (date == null)
        {
            date = new Date();
        }
        if (format == null)
        {
            format = DEFAULT_DATE_TIME_FORMAT;
        }
        SimpleDateFormat df = new SimpleDateFormat(format);
        return df.format(date);
    }

    /**
     * 格式化日期,返回格式为 yyyy-MM-dd
     *
     * @param date 日期
     * @return
     */
    public static String formatAsDate(LocalDateTime date)
    {
        return format(date, DEFAULT_DATE_FORMAT);
    }

    public static String formatAsDateEn(LocalDateTime date)
    {
        return format(date, DEFAULT_DATE_FORMAT_EN);
    }


```

```
168     public static String formatAsYearMonth(LocalDateTime date)
169     {
170         return format(date, DEFAULT_MONTH_FORMAT);
171     }
172
173     public static String formatAsYearMonthEn(LocalDateTime date)
174     {
175         return format(date, DEFAULT_MONTH_FORMAT_EN);
176     }
177
178     /**
179      * 格式化日期,返回格式为 yyyy-ww
180      *
181      * @param date 日期
182      * @return
183      */
184     public static String formatAsYearWeek(LocalDateTime date)
185     {
186         return format(date, DEFAULT_WEEK_FORMAT);
187     }
188
189     public static String formatAsYearWeekEn(LocalDateTime date)
190     {
191         return format(date, DEFAULT_WEEK_FORMAT_EN);
192     }
193
194     /**
195      * 格式化日期,返回格式为 yyyy-MM
196      *
197      * @param date 日期
198      * @return
199      */
200     public static String formatAsYearMonth(Date date)
201     {
202         SimpleDateFormat df = new SimpleDateFormat(DEFAULT_MONTH_FORMAT);
203         return df.format(date);
204     }
205
206     /**
207      * 格式化日期,返回格式为 yyyy-ww
208      *
209      * @param date 日期
210      * @return
211      */
212     public static String formatAsYearWeek(Date date)
213     {
214         SimpleDateFormat df = new SimpleDateFormat(DEFAULT_WEEK_FORMAT);
215         return df.format(date);
216     }
217
218     /**
219      * 格式化日期,返回格式为 HH:mm:ss 例:12:24:24
220      *
221      * @param date 日期
222      * @return
223      */
224     public static String formatAsTime(Date date)
225     {
```

```
226         SimpleDateFormat df = new SimpleDateFormat(DEFAULT_TIME_FORMAT);
227         return df.format(date);
228     }
229 
230     /**
231      * 格式化日期,返回格式为 yyyy-MM-dd
232      *
233      * @param date 日期
234      * @return
235      */
236     public static String formatAsDate(Date date)
237     {
238         SimpleDateFormat df = new SimpleDateFormat(DEFAULT_DATE_FORMAT);
239         return df.format(date);
240     }
241 
242     /**
243      * 格式化日期,返回格式为 yyyy-MM-dd HH:mm:ss
244      *
245      * @param date 日期
246      * @return
247      */
248     public static String formatAsDateTime(Date date)
249     {
250         SimpleDateFormat df = new
SimpleDateFormat(DEFAULT_DATE_TIME_FORMAT);
251         return df.format(date);
252     }
253 
254     /**
255      * 格式化日期,返回格式为 dd ,即对应的天数.
256      *
257      * @param date 日期
258      * @return
259      */
260     public static String formatAsDay(Date date)
261     {
262         SimpleDateFormat df = new SimpleDateFormat("dd");
263         return df.format(date);
264     }
265 
266     //--格式化日期end---------------------------------------
267 
268     //--解析日期start---------------------------------------
269 
270     /**
271      * 将字符转换成日期
272      *
273      * @param dateStr
274      * @param format
275      * @return
276      */
277     public static Date parse(String dateStr, String format)
278     {
279         Date date = null;
280         SimpleDateFormat sdateFormat = new SimpleDateFormat(format);
281         sdateFormat.setLenient(false);
282         try
```

```
283         {
284             date = sdateFormat.parse(dateStr);
285 
286         }
287         catch (Exception e)
288         {
289             log.info("DateUtils error {} ", e);
290         }
291         return date;
292     }
293 
294     /**
295      * 根据传入的String返回对应的date
296      *
297      * @param dateString
298      * @return
299      */
300     public static Date parseAsDate(String dateString)
301     {
302         SimpleDateFormat df = new SimpleDateFormat(DEFAULT_DATE_FORMAT);
303         try
304         {
305             return df.parse(dateString);
306         }
307         catch (ParseException e)
308         {
309             return new Date();
310         }
311     }
312 
313     /**
314      * 按给定参数返回Date对象
315      *
316      * @param dateTime 时间对象格式为("yyyy-MM-dd HH:mm:ss");
317      * @return
318      */
319     public static Date parseAsDateTime(String dateTime)
320     {
321         SimpleDateFormat simpledateformat = new
SimpleDateFormat(DEFAULT_DATE_TIME_FORMAT);
322         try
323         {
324             return simpledateformat.parse(dateTime);
325         }
326         catch (ParseException e)
327         {
328             return null;
329         }
330     }
331 
332     /**
333      * 获取指定日期的开始时间
334      * 如：00:00:00
335      *
336      * @param value
337      * @return
338      */
339     public static Date getDate0000(LocalDateTime value)
```

```
340     {
341         return getDate0000(value.toLocalDate());
342     }
343 
344     /**
345      * 获取指定日期的开始时间
346      * 如：00:00:00
347      *
348      * @param value
349      * @return
350      */
351     public static Date getDate0000(Date value)
352     {
353         return getDate0000(DateUtils.date2LocalDate(value));
354     }
355 
356     /**
357      * 获取指定日期的开始时间
358      * 如：00:00:00
359      *
360      * @param value
361      * @return
362      */
363     public static Date getDate0000(LocalDate value)
364     {
365         LocalDateTime todayStart = LocalDateTime.of(value, LocalTime.MIN);
366         return DateUtils.localDateTime2Date(todayStart);
367     }
368 
369     /**
370      * 获取指定日期的结束时间
371      * 如：23:59:59
372      *
373      * @param value
374      * @return
375      */
376     public static Date getDate2359(LocalDateTime value)
377     {
378         return getDate2359(value.toLocalDate());
379 
380     }
381 
382     /**
383      * 获取指定日期的结束时间
384      * 如：23:59:59
385      *
386      * @param value
387      * @return
388      */
389     public static Date getDate2359(Date value)
390     {
391         return getDate2359(DateUtils.date2LocalDate(value));
392     }
393 
394     /**
395      * 获取指定日期的结束时间
396      * 如：23:59:59
397      *
```

```
398          * @param value
399          * @return
400          */
401         public static Date getDate2359(LocalDate value)
402         {
403             LocalDateTime dateEnd = LocalDateTime.of(value, LocalTime.MAX);
404             return DateUtils.localDateTime2Date(dateEnd);
405         }
406 
407         /**
408          * LocalDateTime转换为Date
409          *
410          * @param localDateTime
411          */
412         public static Date localDateTime2Date(LocalDateTime localDateTime)
413         {
414             ZoneId zoneId = ZoneId.systemDefault();
415             ZonedDateTime zdt = localDateTime.atZone(zoneId);
416             return Date.from(zdt.toInstant());
417         }
418 
419         //--解析日期 end----------------------------------------
420 
421 
422         /**
423          * Date转换为LocalDateTime
424          *
425          * @param date
426          */
427         public static LocalDateTime date2LocalDateTime(Date date)
428         {
429             if (date == null)
430             {
431                 return LocalDateTime.now();
432             }
433             Instant instant = date.toInstant();
434             ZoneId zoneId = ZoneId.systemDefault();
435             return instant.atZone(zoneId).toLocalDateTime();
436         }
437 
438         /**
439          * 日期转 LocalDate
440          *
441          * @param date
442          * @return
443          */
444         public static LocalDate date2LocalDate(Date date)
445         {
446             if (date == null)
447             {
448                 return LocalDate.now();
449             }
450             Instant instant = date.toInstant();
451             ZoneId zoneId = ZoneId.systemDefault();
452             return instant.atZone(zoneId).toLocalDate();
453         }
454 
455         /**
```

```
     * 日期转 LocalTime
     *
     * @param date
     * @return
     */
    public static LocalTime date2LocalTime(Date date)
    {
        if (date == null)
        {
            return LocalTime.now();
        }
        Instant instant = date.toInstant();
        ZoneId zoneId = ZoneId.systemDefault();
        return instant.atZone(zoneId).toLocalTime();
    }

    //-计算日期 start----------------------------------------


    /**
     * 计算结束时间与当前时间中的天数
     *
     * @param endDate 结束日期
     * @return
     */
    public static long until(Date endDate)
    {
        return LocalDateTime.now().until(date2LocalDateTime(endDate),
ChronoUnit.DAYS);
    }

    /**
     * 计算结束时间与开始时间中的天数
     *
     * @param startDate 开始日期
     * @param endDate   结束日期
     * @return
     */
    public static long until(Date startDate, Date endDate)
    {
        return
date2LocalDateTime(startDate).until(date2LocalDateTime(endDate),
ChronoUnit.DAYS);
    }


    /**
     * 计算结束时间与开始时间中的天数
     *
     * @param startDate 开始日期
     * @param endDate   结束日期
     * @return
     */
    public static long until(LocalDateTime startDate, LocalDateTime
endDate)
    {
        return startDate.until(endDate, ChronoUnit.DAYS);
    }
```

```
510
511     public static long until(LocalDate startDate, LocalDate endDate)
512     {
513         return startDate.until(endDate, ChronoUnit.DAYS);
514     }
515
516     /**
517      * 计算2个日期之间的所有的日期 yyyy-MM-dd
518      * 含头含尾
519      *
520      * @param start yyyy-MM-dd
521      * @param end   yyyy-MM-dd
522      * @return
523      */
524     public static List<String> getBetweenDay(Date start, Date end)
525     {
526         return getBetweenDay(date2LocalDate(start), date2LocalDate(end));
527     }
528
529     /**
530      * 计算2个日期之间的所有的日期 yyyy-MM-dd
531      * 含头含尾
532      *
533      * @param start yyyy-MM-dd
534      * @param end   yyyy-MM-dd
535      * @return
536      */
537     public static List<String> getBetweenDay(String start, String end)
538     {
539         return getBetweenDay(LocalDate.parse(start), LocalDate.parse(end));
540     }
541
542     /**
543      * 计算2个日期之间的所有的日期 yyyy-MM-dd
544      * 含头含尾
545      *
546      * @param startDate yyyy-MM-dd
547      * @param endDate   yyyy-MM-dd
548      * @return
549      */
550     public static List<String> getBetweenDay(LocalDate startDate, LocalDate
endDate)
551     {
552         return getBetweenDay(startDate, endDate, DEFAULT_DATE_FORMAT);
553     }
554
555     public static List<String> getBetweenDayEn(LocalDate startDate,
LocalDate endDate)
556     {
557         return getBetweenDay(startDate, endDate, DEFAULT_DATE_FORMAT_EN);
558     }
559
560     public static List<String> getBetweenDay(LocalDate startDate, LocalDate
endDate, String pattern)
561     {
562         if (pattern == null)
563         {
564             pattern = DEFAULT_DATE_FORMAT;
```

```
        }
        List<String> list = new ArrayList<>();
        long distance = ChronoUnit.DAYS.between(startDate, endDate);
        if (distance < 1)
        {
            return list;
        }
        String finalPattern = pattern;
        Stream.iterate(startDate, d -> d.plusDays(1)).
                limit(distance + 1)
                .forEach(f ->
list.add(f.format(DateTimeFormatter.ofPattern(finalPattern))));
        return list;
    }


    /**
     * 计算2个日期之间的所有的周 yyyy-ww
     * 含头含尾
     *
     * @param start yyyy-MM-dd
     * @param end   yyyy-MM-dd
     * @return
     */
    public static List<String> getBetweenWeek(Date start, Date end)
    {
        return getBetweenWeek(date2LocalDate(start), date2LocalDate(end));
    }

    /**
     * 计算2个日期之间的所有的周 yyyy-ww
     * 含头含尾
     *
     * @param start yyyy-MM-dd
     * @param end   yyyy-MM-dd
     * @return
     */
    public static List<String> getBetweenWeek(String start, String end)
    {
        return getBetweenWeek(LocalDate.parse(start),
LocalDate.parse(end));
    }

    /**
     * 计算2个日期之间的所有的周 yyyy-ww
     * 含头含尾
     *
     * @param startDate yyyy-MM-dd
     * @param endDate   yyyy-MM-dd
     * @return
     */
    public static List<String> getBetweenWeek(LocalDate startDate,
LocalDate endDate)
    {
        return getBetweenWeek(startDate, endDate, DEFAULT_WEEK_FORMAT);
    }

```

```
619     public static List<String> getBetweenWeek(LocalDate startDate,
    LocalDate endDate, String pattern)
620     {
621         List<String> list = new ArrayList<>();
622 
623         long distance = ChronoUnit.WEEKS.between(startDate, endDate);
624         if (distance < 1)
625         {
626             return list;
627         }
628         Stream.iterate(startDate, d -> d.plusWeeks(1)).
629                 limit(distance + 1).forEach(f ->
    list.add(f.format(DateTimeFormatter.ofPattern(pattern))));
630         return list;
631     }
632 
633     /**
634      * 计算2个日期之间的所有的月 yyyy-MM
635      *
636      * @param start yyyy-MM-dd
637      * @param end   yyyy-MM-dd
638      * @return
639      */
640     public static List<String> getBetweenMonth(Date start, Date end)
641     {
642         return getBetweenMonth(date2LocalDate(start), date2LocalDate(end));
643     }
644 
645     /**
646      * 计算2个日期之间的所有的月 yyyy-MM
647      *
648      * @param start yyyy-MM-dd
649      * @param end   yyyy-MM-dd
650      * @return
651      */
652     public static List<String> getBetweenMonth(String start, String end)
653     {
654         return getBetweenMonth(LocalDate.parse(start),
    LocalDate.parse(end));
655     }
656 
657     /**
658      * 计算2个日期之间的所有的月 yyyy-MM
659      *
660      * @param startDate yyyy-MM-dd
661      * @param endDate   yyyy-MM-dd
662      * @return
663      */
664     public static List<String> getBetweenMonth(LocalDate startDate,
    LocalDate endDate)
665     {
666         return getBetweenMonth(startDate, endDate, DEFAULT_MONTH_FORMAT);
667     }
668 
669     public static List<String> getBetweenMonth(LocalDate startDate,
    LocalDate endDate, String pattern)
670     {
671         List<String> list = new ArrayList<>();
```

```
672         long distance = ChronoUnit.MONTHS.between(startDate, endDate);
673         if (distance < 1)
674         {
675             return list;
676         }
677 
678         Stream.iterate(startDate, d -> d.plusMonths(1))
679                 .limit(distance + 1)
680                 .forEach(f ->
list.add(f.format(DateTimeFormatter.ofPattern(pattern))));
681         return list;
682     }
683 
684     /**
685      * 计算时间区间内的日期列表，并返回
686      *
687      * @param startTime
688      * @param endTime
689      * @param dateList
690      * @return
691      */
692     public static String calculationEn(LocalDateTime startTime,
LocalDateTime endTime, List<String> dateList)
693     {
694         if (startTime == null)
695         {
696             startTime = LocalDateTime.now();
697         }
698         if (endTime == null)
699         {
700             endTime = LocalDateTime.now().plusDays(30);
701         }
702         return calculationEn(startTime.toLocalDate(),
endTime.toLocalDate(), dateList);
703     }
704 
705     public static String calculation(LocalDate startDate, LocalDate
endDate, List<String> dateList)
706     {
707         if (startDate == null)
708         {
709             startDate = LocalDate.now();
710         }
711         if (endDate == null)
712         {
713             endDate = LocalDate.now().plusDays(30);
714         }
715         if (dateList == null)
716         {
717             dateList = new ArrayList<>();
718         }
719         long day = until(startDate, endDate);
720 
721         String dateType = MONTH;
722         if (day >= 0 && day <= MAX_MONTH_DAY)
723         {
724             dateType = DAY;
```

```
            dateList.addAll(DateUtils.getBetweenDay(startDate, endDate,
DEFAULT_DATE_FORMAT));
        }
        else if (day > MAX_MONTH_DAY && day <= MAX_3_MONTH_DAY)
        {
            dateType = WEEK;
            dateList.addAll(DateUtils.getBetweenWeek(startDate, endDate,
DEFAULT_WEEK_FORMAT));
        }
        else if (day > MAX_3_MONTH_DAY && day <= MAX_YEAR_DAY)
        {
            dateType = MONTH;
            dateList.addAll(DateUtils.getBetweenMonth(startDate, endDate,
DEFAULT_MONTH_FORMAT));
        }
        else
        {
            throw new BizException("日期参数只能介于0-365天之间");
        }
        return dateType;
    }

    public static String calculationEn(LocalDate startDate, LocalDate
endDate, List<String> dateList)
    {
        if (startDate == null)
        {
            startDate = LocalDate.now();
        }
        if (endDate == null)
        {
            endDate = LocalDate.now().plusDays(30);
        }
        if (dateList == null)
        {
            dateList = new ArrayList<>();
        }
        long day = until(startDate, endDate);

        String dateType = MONTH;
        if (day >= 0 && day <= MAX_MONTH_DAY)
        {
            dateType = DAY;
            dateList.addAll(DateUtils.getBetweenDay(startDate, endDate,
DEFAULT_DATE_FORMAT_EN));
        }
        else if (day > MAX_MONTH_DAY && day <= MAX_3_MONTH_DAY)
        {
            dateType = WEEK;
            dateList.addAll(DateUtils.getBetweenWeek(startDate, endDate,
DEFAULT_WEEK_FORMAT_EN));
        }
        else if (day > MAX_3_MONTH_DAY && day <= MAX_YEAR_DAY)
        {
            dateType = MONTH;
            dateList.addAll(DateUtils.getBetweenMonth(startDate, endDate,
DEFAULT_MONTH_FORMAT_EN));
        }
```

```
        else
        {
            throw new BizException("日期参数只能介于0-365天之间");
        }
        return dateType;
    }

//----------//----------//----------//----------//----------//----------//----------//----------//----------//----------

}
```

第十五步：编写工具类JwtHelper

```
package com.example.tools_jwt.utils;

import com.example.tools_jwt.constants.BaseContextConstants;
import com.example.tools_jwt.entity.JwtUserInfo;
import com.example.tools_jwt.entity.Token;
import com.example.tools_jwt.exception.BizException;
import com.example.tools_jwt.exception.ExceptionCode;
import io.jsonwebtoken.*;

import java.io.IOException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.time.LocalDateTime;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.ExpiredJwtException;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.SignatureException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.utils
 * Class(类名): JwtHelper
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/2
 * Time(创建时间)： 22:24
 * Version(版本): 1.0
 * Description(描述)： 无
 */
```

```
37
38  public class JwtHelper
39  {
40      private static final RsaKeyHelper RSA_KEY_HELPER = new RsaKeyHelper();
41
42      private static final Logger log =
    LoggerFactory.getLogger(JwtHelper.class);
43
44      /**
45       * 生成用户令牌
46       *
47       * @param jwtInfo    jwt信息
48       * @param priKeyPath 私钥路径
49       * @param expire     到期时间
50       * @return {@link Token}
51       * @throws BizException 业务异常
52       */
53      public static Token generateUserToken(JwtUserInfo jwtInfo, String
    priKeyPath, int expire) throws BizException
54      {
55          JwtBuilder jwtBuilder = Jwts.builder()
56                  //设置主题
57                  .setSubject(String.valueOf(jwtInfo.getUserId()))
58                  .claim(BaseContextConstants.JWT_KEY_ACCOUNT,
    jwtInfo.getAccount())
59                  .claim(BaseContextConstants.JWT_KEY_NAME,
    jwtInfo.getName())
60                  .claim(BaseContextConstants.JWT_KEY_ORG_ID,
    jwtInfo.getOrgId())
61                  .claim(BaseContextConstants.JWT_KEY_STATION_ID,
    jwtInfo.getStationId());
62          return generateToken(jwtBuilder, priKeyPath, expire);
63      }
64
65
66      /**
67       * 获取token中的用户信息
68       *
69       * @param token      令牌
70       * @param pubKeyPath 公钥路径
71       * @return {@link JwtUserInfo}
72       * @throws BizException 业务异常
73       */
74      public static JwtUserInfo getJwtFromToken(String token, String
    pubKeyPath) throws BizException
75      {
76          Jws<Claims> claimsJws = parserToken(token, pubKeyPath);
77          Claims body = claimsJws.getBody();
78          String strUserId = body.getSubject();
79          String account =
    StrHelper.getObjectValue(body.get(BaseContextConstants.JWT_KEY_ACCOUNT));
80          String name =
    StrHelper.getObjectValue(body.get(BaseContextConstants.JWT_KEY_NAME));
81          String strOrgId =
    StrHelper.getObjectValue(body.get(BaseContextConstants.JWT_KEY_ORG_ID));
82          String strDepartmentId =
    StrHelper.getObjectValue(body.get(BaseContextConstants.JWT_KEY_STATION_ID))
    ;
```

```
        Long userId = NumberHelper.longValueOf0(strUserId);
        Long orgId = NumberHelper.longValueOf0(strOrgId);
        Long departmentId = NumberHelper.longValueOf0(strDepartmentId);
        return new JwtUserInfo(userId, account, name, orgId, departmentId);
    }


    /**
     * 生成token
     *
     * @param builder    构建器
     * @param priKeyPath 私钥路径
     * @param expire     到期时间
     * @return {@link Token}
     * @throws BizException 业务异常
     */
    protected static Token generateToken(JwtBuilder builder, String priKeyPath, int expire) throws BizException
    {
        try
        {
            //返回的字符串便是我们的jwt串了
            String compactJws = builder.setExpiration(DateUtils.localDateTime2Date(LocalDateTime.now().plusSeconds(expire)))
                    //设置算法（必须）
                    .signWith(SignatureAlgorithm.RS256, RSA_KEY_HELPER.getPrivateKey(priKeyPath))
                    //这个是全部设置完成后拼成jwt串的方法
                    .compact();
            return new Token(compactJws, expire);
        }
        catch (IOException | NoSuchAlgorithmException | InvalidKeySpecException e)
        {
            log.error("errcode:{}, message:{}", ExceptionCode.JWT_GEN_TOKEN_FAIL.getCode(), e.getMessage());
            throw new BizException(ExceptionCode.JWT_GEN_TOKEN_FAIL.getCode(), ExceptionCode.JWT_GEN_TOKEN_FAIL.getMsg());
        }
    }


    /**
     * 公钥解析token
     *
     * @param token      令牌
     * @param pubKeyPath 公钥路径
     * @return {@link Jws}<{@link Claims}>
     * @throws BizException 业务异常
     */
    private static Jws<Claims> parserToken(String token, String pubKeyPath) throws BizException
    {
        try
        {
```

```
            return
Jwts.parser().setSigningKey(RSA_KEY_HELPER.getPublicKey(pubKeyPath)).parseC
laimsJws(token);
        }
        catch (ExpiredJwtException ex)
        {
            //过期
            throw new
BizException(ExceptionCode.JWT_TOKEN_EXPIRED.getCode(),
ExceptionCode.JWT_TOKEN_EXPIRED.getMsg());
        }
        catch (SignatureException ex)
        {
            //签名错误
            throw new BizException(ExceptionCode.JWT_SIGNATURE.getCode(),
ExceptionCode.JWT_SIGNATURE.getMsg());
        }
        catch (IllegalArgumentException ex)
        {
            //token 为空
            throw new
BizException(ExceptionCode.JWT_ILLEGAL_ARGUMENT.getCode(),
ExceptionCode.JWT_ILLEGAL_ARGUMENT.getMsg());
        }
        catch (Exception e)
        {
            log.error("errcode:{}, message:{}",
ExceptionCode.JWT_PARSER_TOKEN_FAIL.getCode(), e.getMessage());
            throw new
BizException(ExceptionCode.JWT_PARSER_TOKEN_FAIL.getCode(),
ExceptionCode.JWT_PARSER_TOKEN_FAIL.getMsg());
        }
    }
}
```

第十六步：编写类AuthClientConfigurationProperties

```
package com.example.tools_jwt.client.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

import static
com.example.tools_jwt.client.config.AuthClientConfigurationProperties.PREFIX
;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.client.config
```

```
 * Class(类名): AuthClientConfigurationProperties
 * Author(作者）: mao
 * Author QQ: 1296193245
 * GitHub: https://github.com/maomao124/
 * Date(创建日期): 2022/11/3
 * Time(创建时间): 12:42
 * Version(版本): 1.0
 * Description(描述): 无
 */

@ConfigurationProperties(prefix = PREFIX)
public class AuthClientConfigurationProperties
{
    public static final String PREFIX = "authentication";

    private TokenInfo user;

    public AuthClientConfigurationProperties()
    {

    }

    public TokenInfo getUser()
    {
        return user;
    }

    public void setUser(TokenInfo user)
    {
        this.user = user;
    }

    public static class TokenInfo
    {
        /**
         * 请求头名称
         */
        private String headerName;
        /**
         * 解密 网关使用
         */
        private String pubKey;

        public TokenInfo()
        {

        }

        public TokenInfo(String headerName, String pubKey)
        {
            this.headerName = headerName;
            this.pubKey = pubKey;
        }

        public String getHeaderName()
        {
            return headerName;
        }
```

```
68
69          public void setHeaderName(String headerName)
70          {
71              this.headerName = headerName;
72          }
73
74          public String getPubKey()
75          {
76              return pubKey;
77          }
78
79          public void setPubKey(String pubKey)
80          {
81              this.pubKey = pubKey;
82          }
83      }
84  }
```

第十七步：编写工具类JwtTokenClientUtils

```
1  package com.example.tools_jwt.client.utils;
2
3  import
   com.example.tools_jwt.client.config.AuthClientConfigurationProperties;
4  import com.example.tools_jwt.entity.JwtUserInfo;
5  import com.example.tools_jwt.exception.BizException;
6  import com.example.tools_jwt.utils.JwtHelper;
7
8  /**
9   * Project name(项目名称)：jwt_spring_boot_starter
10  * Package(包名): com.example.tools_jwt.client.utils
11  * Class(类名): JwtTokenClientUtils
12  * Author(作者）: mao
13  * Author QQ：1296193245
14  * GitHub：https://github.com/maomao124/
15  * Date(创建日期)： 2022/11/3
16  * Time(创建时间)： 12:45
17  * Version(版本): 1.0
18  * Description(描述)： 无
19  */
20
21 public class JwtTokenClientUtils
22 {
23     /**
24      * 用于 认证服务的 客户端使用（如 网关） ， 在网关获取到token后，
25      * 调用此工具类进行token 解析。
26      * 客户端一般只需要解析token 即可
27      */
28     private final AuthClientConfigurationProperties
    authClientConfigurationProperties;
```

```
29
30      public JwtTokenClientUtils(AuthClientConfigurationProperties
        authClientConfigurationProperties)
31      {
32          this.authClientConfigurationProperties =
        authClientConfigurationProperties;
33      }
34
35      /**
36       * 解析token，获取用户信息
37       *
38       * @param token 令牌
39       * @return {@link JwtUserInfo}
40       * @throws BizException 业务异常
41       */
42      public JwtUserInfo getUserInfo(String token) throws BizException
43      {
44          AuthClientConfigurationProperties.TokenInfo userTokenInfo =
        authClientConfigurationProperties.getUser();
45          return JwtHelper.getJwtFromToken(token, userTokenInfo.getPubKey());
46      }
47  }
```

第十八步：编写配置类AuthClientConfiguration

```
1   package com.example.tools_jwt.client.config;
2
3   import com.example.tools_jwt.client.utils.JwtTokenClientUtils;
4   import org.springframework.beans.factory.annotation.Autowired;
5   import
    org.springframework.boot.context.properties.EnableConfigurationProperties;
6   import org.springframework.context.annotation.Bean;
7
8   /**
9    * Project name(项目名称)：jwt_spring_boot_starter
10   * Package(包名): com.example.tools_jwt.client.config
11   * Class(类名): AuthClientConfiguration
12   * Author(作者）: mao
13   * Author QQ：1296193245
14   * GitHub：https://github.com/maomao124/
15   * Date(创建日期)： 2022/11/3
16   * Time(创建时间)： 12:49
17   * Version(版本): 1.0
18   * Description(描述)： 无
19   */
20
21  @EnableConfigurationProperties(AuthClientConfigurationProperties.class)
22  public class AuthClientConfiguration
23  {
24      @Bean
```

```
    public JwtTokenClientUtils jwtTokenClientUtils(@Autowired
AuthClientConfigurationProperties authClientConfigurationProperties)
    {
        return new JwtTokenClientUtils(authClientConfigurationProperties);
    }
}
```

第十九步：编写注解EnableAuthClient

```
package com.example.tools_jwt.client;


import com.example.tools_jwt.client.config.AuthClientConfiguration;
import org.springframework.context.annotation.Import;

import java.lang.annotation.*;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@Import(AuthClientConfiguration.class)
public @interface EnableAuthClient
{

/*
配置文件示例：


authentication:
  user:
    # 过期时间
    expire: 1800
    # 公钥位置
    pubKey: keys/pub.key

 */
}

```

第二十步：编写类AuthServerConfigurationProperties

```java
package com.example.tools_jwt.server.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.server.config
 * Class(类名): AuthServerConfigurationProperties
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/3
 * Time(创建时间)： 12:55
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@ConfigurationProperties(prefix = AuthServerConfigurationProperties.PREFIX)
public class AuthServerConfigurationProperties
{
    public static final String PREFIX = "authentication";

    private TokenInfo user;

    public AuthServerConfigurationProperties()
    {

    }

    public AuthServerConfigurationProperties(TokenInfo user)
    {
        this.user = user;
    }

    public TokenInfo getUser()
    {
        return user;
    }

    public void setUser(TokenInfo user)
    {
        this.user = user;
    }

    public static class TokenInfo
    {
        /**
         * 过期时间
         */
        private Integer expire = 7200;
        /**
         * 加密 服务使用
         */
```

```
54          private String priKey;
55          /**
56           * 解密
57           */
58          private String pubKey;
59
60          public TokenInfo()
61          {
62
63          }
64
65          public TokenInfo(Integer expire, String priKey, String pubKey)
66          {
67              this.expire = expire;
68              this.priKey = priKey;
69              this.pubKey = pubKey;
70          }
71
72          public Integer getExpire()
73          {
74              return expire;
75          }
76
77          public void setExpire(Integer expire)
78          {
79              this.expire = expire;
80          }
81
82          public String getPriKey()
83          {
84              return priKey;
85          }
86
87          public void setPriKey(String priKey)
88          {
89              this.priKey = priKey;
90          }
91
92          public String getPubKey()
93          {
94              return pubKey;
95          }
96
97          public void setPubKey(String pubKey)
98          {
99              this.pubKey = pubKey;
100         }
101     }
102 }
```

第二十一步：编写工具类JwtTokenServerUtils

```
package com.example.tools_jwt.server.utils;

import com.example.tools_jwt.entity.JwtUserInfo;
import com.example.tools_jwt.entity.Token;
import com.example.tools_jwt.exception.BizException;
import
com.example.tools_jwt.server.config.AuthServerConfigurationProperties;
import com.example.tools_jwt.utils.JwtHelper;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.server.utils
 * Class(类名): JwtTokenServerUtils
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/3
 * Time(创建时间)： 12:58
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class JwtTokenServerUtils
{
    /**
     * 认证服务端使用，如 authority-server
     * 生成和 解析token
     */
    private final AuthServerConfigurationProperties
authServerConfigurationProperties;

    public JwtTokenServerUtils(AuthServerConfigurationProperties
authServerConfigurationProperties)
    {
        this.authServerConfigurationProperties =
authServerConfigurationProperties;
    }


    /**
     * 生成token
     *
     * @param jwtInfo jwt信息
     * @param expire  到期时间
     * @return {@link Token}
     * @throws BizException 业务异常
     */
    public Token generateUserToken(JwtUserInfo jwtInfo, Integer expire)
throws BizException
    {
        AuthServerConfigurationProperties.TokenInfo userTokenInfo =
authServerConfigurationProperties.getUser();
        if (expire == null || expire <= 0)
```

```
        {
            expire = userTokenInfo.getExpire();
        }
        return JwtHelper.generateUserToken(jwtInfo,
userTokenInfo.getPriKey(), expire);
    }


    /**
     * 解析token
     *
     * @param token 令牌
     * @return {@link JwtUserInfo}
     * @throws BizException 业务异常
     */
    public JwtUserInfo getUserInfo(String token) throws BizException
    {
        AuthServerConfigurationProperties.TokenInfo userTokenInfo =
authServerConfigurationProperties.getUser();
        return JwtHelper.getJwtFromToken(token, userTokenInfo.getPubKey());
    }
}
```

第二十二步：编写工具类AuthServerConfiguration

```
package com.example.tools_jwt.server.config;

import com.example.tools_jwt.server.utils.JwtTokenServerUtils;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;

/**
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): com.example.tools_jwt.server.config
 * Class(类名): AuthServerConfiguration
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/3
 * Time(创建时间)： 13:02
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@EnableConfigurationProperties(AuthServerConfigurationProperties.class)
public class AuthServerConfiguration
{
    @Bean
```

```
    public JwtTokenServerUtils jwtTokenServerUtils(@Autowired
AuthServerConfigurationProperties authServerConfigurationProperties)
    {
        return new JwtTokenServerUtils(authServerConfigurationProperties);
    }
}
```

第二十三步：编写注解EnableAuthServer

```
package com.example.tools_jwt.server;

import com.example.tools_jwt.server.config.AuthServerConfiguration;
import org.springframework.context.annotation.Import;

import java.lang.annotation.*;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@Import(AuthServerConfiguration.class)
public @interface EnableAuthServer
{

/*

配置文件示例：


authentication:
  user:
    # 过期时间
    expire: 1800
    # 私钥位置
    prikey: keys/pri.key
    # 公钥位置
    pubKey: keys/pub.key

 */
}

```

使用starter

JwtTokenServerUtils主要是提供给权限服务的，类中包含生成jwt和解析jwt两个方法

JwtTokenClientUtils主要是提供给网关服务的，类中只有一个解析jwt的方法

需要注意的是tools-jwt并不是starter，所以如果只是在项目中引入他的maven坐标并不能直接使用其提供的工具类。需要在启动类上加入pd-tools-jwt模块中定义的注解@EnableAuthServer或者@EnableAuthClient。

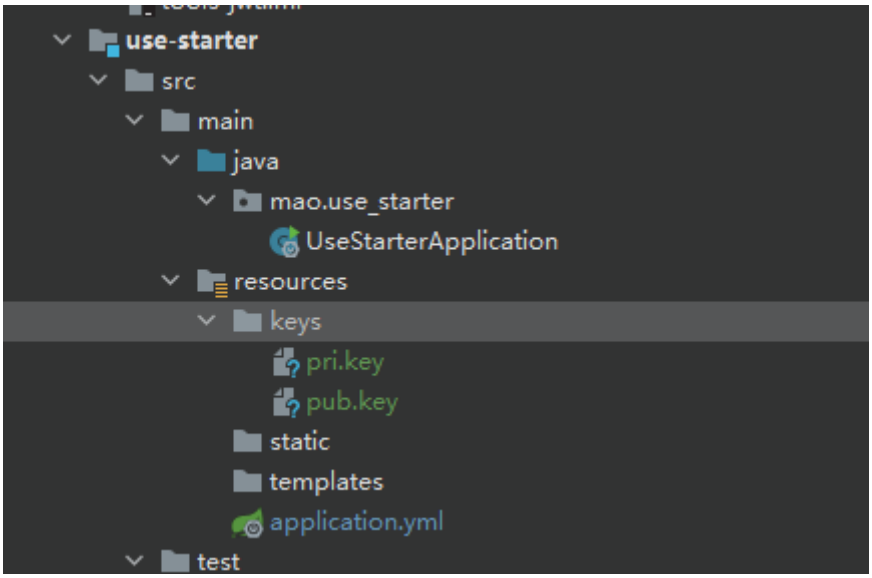
tools-jwt使用的签名算法为RS256，需要我们自己的应用来提供一对公钥和私钥，然后在application.yml中进行配置即可

第一步：导入tools-jwt的依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <artifactId>jwt_spring_boot_starter</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>use-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>use-starter</name>
    <description>use-starter</description>
    <properties>

    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>mao</groupId>
            <artifactId>tools-jwt</artifactId>
```

```
            <version>0.0.1-SNAPSHOT</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第二步：在资源路径下创建keys目录，将通过RSA算法生成的公钥和私钥复制到此目录下



第三步：编写application.yml文件

```yaml
authentication:
  user:
    # 过期时间
    expire: 1800
    # 私钥位置
    prikey: keys/pri.key
    # 公钥位置
    pubKey: keys/pub.key
```

第四步：在启动类加注解@EnableAuthServer

```java
package mao.use_starter;

import com.example.tools_jwt.server.EnableAuthServer;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@EnableAuthServer
public class UseStarterApplication
{

    public static void main(String[] args)
    {
        SpringApplication.run(UseStarterApplication.class, args);
    }

}
```

第五步：编写UserController

```java
package mao.use_starter.controller;

import com.example.tools_jwt.entity.JwtUserInfo;
import com.example.tools_jwt.entity.Token;
import com.example.tools_jwt.exception.BizException;
import com.example.tools_jwt.server.utils.JwtTokenServerUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

/**
```

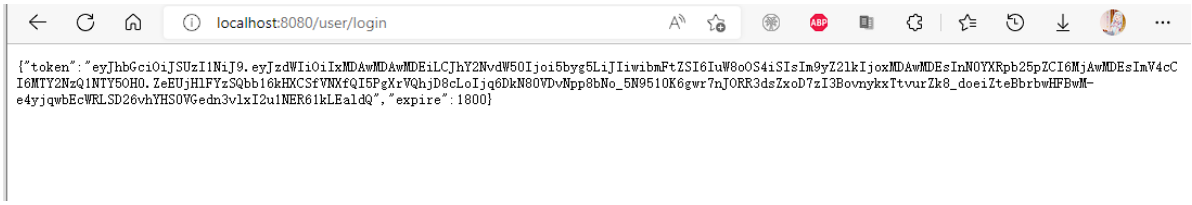
```
 * Project name(项目名称)：jwt_spring_boot_starter
 * Package(包名): mao.use_starter.controller
 * Class(类名): UserController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/3
 * Time(创建时间)： 13:24
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
@RequestMapping("/user")
public class UserController
{

    @Autowired
    private JwtTokenServerUtils jwtTokenServerUtils;

    private static final Logger log = LoggerFactory.getLogger(UserController.class);

    @GetMapping("/login")
    public Token login()
    {
        JwtUserInfo jwtUserInfo = new JwtUserInfo();
        jwtUserInfo.setName("张三");
        jwtUserInfo.setOrgId(100001L);
        jwtUserInfo.setUserId(100000001L);
        jwtUserInfo.setAccount("张三");
        jwtUserInfo.setStationId(20001L);
        Token token = jwtTokenServerUtils.generateUserToken(jwtUserInfo, null);
        log.info(token.getToken());
        return token;
    }

    @GetMapping("/{token}")
    public boolean test(@PathVariable String token)
    {
        log.info(token);

        try
        {
            JwtUserInfo userInfo = jwtTokenServerUtils.getUserInfo(token);
            log.info(userInfo.toString());
            return true;
        }
        catch (BizException e)
        {
            log.error(e.getMessage());
            return false;
        }
    }
}
```

第六步：启动服务

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-11-03 13:37:33.952  INFO 1868 --- [           main] mao.use_starter.UseStarterApplication    : Starting UseStarterApplication using Java 16.0.2 on mao with PID 1868 (H:\程序\大四上期\jwt_spring_boot_starter\use-starter\target\classes started by mao in H:\程序\大四上期\jwt_spring_boot_starter)
2022-11-03 13:37:33.955  INFO 1868 --- [           main] mao.use_starter.UseStarterApplication    : No active profile set, falling back to 1 default profile: "default"
2022-11-03 13:37:34.632  INFO 1868 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-11-03 13:37:34.639  INFO 1868 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-03 13:37:34.640  INFO 1868 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-11-03 13:37:34.730  INFO 1868 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-11-03 13:37:34.730  INFO 1868 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 738 ms
2022-11-03 13:37:34.974  INFO 1868 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-11-03 13:37:34.984  INFO 1868 --- [           main] mao.use_starter.UseStarterApplication    : Started UseStarterApplication in 1.31 seconds (JVM running for 1.774)
```

第七步：访问

http://localhost:8080/user/login



将拿到的token拼接到http://localhost:8080/user/的后面

```
true
```

```
2022-11-03 13:40:25.213  INFO 1868 --- [nio-8080-exec-9] m.use_starter.controller.UserController  : JwtUserInfo{userId=100000001, account='张三', name='张三', orgId=100001, stationId=20001}
```

尝试更改一个字符

```
false
```

```
2022-11-03 13:40:28.529 ERROR 1868 --- [io-8080-exec-10] m.use_starter.controller.UserController  : 不合法的token，请认真比对 token 的签名
```

end

by mao

2022 11 03