JAVA-JWT Documentation

Java-JWT is a library that provides a simple way to create, parse, and validate JSON Web Tokens (JWT) in Java applications. JWT is a compact and URL-safe means of representing claims to be transferred between 2 parties. It is used for authentication and data exchange and their security depends on properly protecting the secret key and verifying the token signatures on the server side.

This document will guide you through the installation, basic usage, and important concepts of java-jwt library.

Table of Contents

1. Installation

To use Java-JWT in your Java project, you can add the library as a dependency using your build tool. For Maven, add the following to your ‘pom.xml’ file:

```
<dependency>
<groupId>com.auth0</groupId>
<artifactId>java-jwt</artifactId>
<version>4.4.0</version>
</dependency>
```

For Gradle, add the following to your ‘build.gradle’ file:

```
implementation 'com.auth0:java-jwt:4.4.0'
```

Make sure to check the official Maven Central or other repositories for the latest version of the library.

2. Creating a JWT

To create a JWT, you need to use the ‘com.auth0.jwt.JWT’ class and the ‘com.auth0.jwt.JWTCreator.Builder’ class to set the claims and sign the token with a secret key or an RSA private key. Here's a basic example

```
import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTCreator;

// Build the JWT with claims
String secretKey = "your-secret-key";
String jwt = JWT.create()
                .withIssuer("your-issuer")
                .withSubject("your-subject")
                .withClaim("customClaim", "custom-value")
                 .sign(Algorithm.HMAC256(secretKey));
```

3. Parsing and Validating a JWT

To parse and validate a JWT, you use the ‘com.auth0.jwt.JWT’ class along with the ‘com.auth0.jwt.interfaces.DecodedJWT’ interface. You'll also need to specify the algorithm used to sign the token (the same algorithm used during token creation). Here's an example of how to validate a JWT:

```
import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.DecodedJWT;

String secretKey = "your-secret-key";
String token = "your-jwt-token";

// Verify and decode the token
JWTVerifier verifier=JWT.require(Algorithm.HMAC256(secretKey))
                                                  .build();
DecodedJWT decodedJWT = verifier.verify(token);

// Access the claims from the decoded JWT
String issuer = decodedJWT.getIssuer();
String subject = decodedJWT.getSubject();
String customClaim = decodedJWT.getClaim("customClaim").asString();
```

The ‘JWT.require()’ method sets up the verification requirements and ‘build().verify(token)’ performs the actual verification. If the JWT signature and claims are valid, a ‘DecodedJWT’ object is returned, which allows you to access the payload and header of the token.

4. JWT Claims

JWT claims are pieces of information added to the token payload. Some common claims include:

- ‘iss’ (Issuer): Identifies the entity that issued the token.
- ‘sub’ (Subject): Identifies the subject of the token, typically the user.
- ‘exp’ (Expiration Time): Indicates the time at which the token will expire.
- ‘nbf’ (Not Before): Indicates the time before which the token cannot be accepted.
- ‘iat‘(Issued At): Indicates the time at which the token was issued.

- ‘jti’ (JWT ID): A unique identifier for the token.

JWT can also be integrated by custom payload and header claims, by using the withHeader and withClaim methods. The statement with set the header of JWT and there should be a map which is to be initialized in the parameter such as the signing algorithm, token type,etc.

Claims can be of various types like String, numbers, Booleans, etc. Another claim of current timestamp can be added for example ‘Instant.now()’ . After adding the desired claims, the ‘sign()’ method is called on the ‘JWT Builder’ object. It will take as an argument to sign the JWT and produce the final token as a String. Same will go with ‘withIsssuer()’ method as it will set the issuer claim in the JWT.

5. JWT Algorithms

Java-JWT supports various cryptographic algorithms to sign and verify tokens. Some common algorithms include:

- ‘Algorithm.HMAC256(secretKey)’: HMAC SHA-256 algorithm using a shared secret key.
- ‘Algorithm.RSA256(publicKey, privateKey)’: RSA SHA-256 algorithm using an RSA public and private key pair.

An algorithm instance is created with RSA256 as signing algorithm, using both the RSA public key and the RSA Private Key as arguments.

If the code is implemented by using a KeyProvider then a JwkProvider is created using the JwkProvider Builder. The jwkProvider is responsible for fetching and caching JSON Web Keys(Jwks) needed for validating JWT signatures. It specifies the URL where the Jwks are available, sets up caching and rate limiting to avoid excessive request to the JWK Provider.

A custom implementation of RSA Key Provider is created. This interface provides methods for obtaining the public key corresponding to a specific key ID, the private key used for signing ,and the ID of private key.

6. Token Expiration and Refreshing

JWTs can have an expiration time (exp claim) set to make them valid for a specific duration. After the token expires, clients need to request a new one. You can issue a refreshed token when the old one is about to expire, usually using the refresh token mechanism.

7. Security Considerations

- Always use a strong and secure secret key or private key for signing tokens.
- Ensure that tokens are transmitted over secure channels (e.g., HTTPS) to prevent interception or tampering.
- Be cautious with including sensitive information in the JWT payload as it can be easily decoded (use encryption for sensitive data).

8.Examples

For more comprehensive examples and advanced usage, refer to the official Java-JWT documentation and the library's GitHub repository: https://github.com/auth0/java-jwt

Please note that this documentation provides a basic overview of the Java-JWT library. For detailed and up-to-date information, refer to the official documentation and the library's repository.