**JAAS**

**Understanding the Role of JAAS in Pluggable/Extensible Authentication for Web Applications**

**🔐 What is JAAS?**

**Java Authentication and Authorization Service (JAAS)** is a **Java security framework** that provides:

* **Authentication**: Who are you?
* **Authorization**: What are you allowed to do?

JAAS is **pluggable**, meaning you can **plug in different login modules** (e.g., LDAP, DB, Kerberos) **without changing application code**.

**🎯 Role of JAAS in Web Applications**

JAAS supports **extensible and customizable authentication** in web applications, particularly useful when:

* You need to use a **custom user store** (like database, LDAP, OAuth).
* You want to **centralize** security logic across multiple applications.
* You need **stronger authentication** than what servlet containers (like Tomcat) offer out-of-the-box.

**🧩 Key JAAS Concepts**

| **Concept** | **Description** |
| --- | --- |
| LoginContext | Manages authentication. You call login() on it. |
| LoginModule | Pluggable logic for authenticating users. Can be reused. |
| Subject | Represents the authenticated user and their principals (roles). |
| Principal | Represents identity (username, roles, etc.). |

**🏗️ How JAAS Enables Pluggable Authentication**

1. **Configure Login Module in jaas.config**:

MyAppLogin {

com.example.auth.MyLoginModule required;

};

1. **Use JAAS Login in Code**:

LoginContext lc = new LoginContext("MyAppLogin", new MyCallbackHandler());

lc.login(); // executes your LoginModule

Subject subject = lc.getSubject();

1. **Use JAAS Principals in Authorization**:  
   Your app can enforce role-based access via the Subject and Principal objects returned by JAAS.

**📦 JAAS in Web Containers (e.g., Tomcat, JBoss)**

**✅ JAAS Integration in Tomcat:**

* Tomcat supports JAAS realms via:

<Realm className="org.apache.catalina.realm.JAASRealm"

appName="MyAppLogin"

userClassNames="com.example.auth.UserPrincipal"

roleClassNames="com.example.auth.RolePrincipal"/>

Tomcat uses JAAS LoginContext to authenticate users and map roles to its container security model.

**🔁 Benefits of JAAS in Web Apps**

| **Benefit** | **Description** |
| --- | --- |
| 🔌 Pluggable | Swap authentication mechanism without rewriting business logic. |
| 🔐 Centralized Security | Manage users/roles outside app code. |
| 🧩 Extensible | You can use LDAP, DB, OAuth, Kerberos, or custom logic. |
| 🔄 Reusable | Same login modules can be used across different apps. |
| 🧠 Fine-grained Control | Full access to user identity and roles after login. |

**🧪 Use Case Examples**

* Enterprise apps using **LDAP + DB fallback**
* SSO across multiple Java EE apps
* Applications needing **stronger authentication** (e.g., 2FA, biometric)

**📝 Summary**

JAAS provides a **pluggable authentication mechanism** that separates authentication logic from application code. In web applications:

* It allows using **custom login modules** (LDAP, DB, etc.).
* JAAS integrates with containers (like Tomcat) via **JAASRealm**.
* It enables **reusable, centralized, and extensible security**.

Here's a complete, working example of **JAAS authentication in Tomcat** using a **custom login module**.

**✅ Overview**

* **Web app name:** JAASWebApp
* **Authentication method:** JAAS (LoginContext)
* **LoginModule:** MyLoginModule
* **Realm:** JAASRealm in server.xml
* **Tomcat version:** 9/10

**🔧 Step-by-Step Setup**

**✅ 1. MyLoginModule.java**

Create your **custom login module**:

package com.example.auth;

import javax.security.auth.\*;

import javax.security.auth.callback.\*;

import javax.security.auth.login.\*;

import javax.security.auth.spi.LoginModule;

import java.util.\*;

public class MyLoginModule implements LoginModule {

private Subject subject;

private CallbackHandler callbackHandler;

private String username;

private List<Principal> principals = new ArrayList<>();

@Override

public void initialize(Subject subject, CallbackHandler callbackHandler,

Map<String, ?> sharedState, Map<String, ?> options) {

this.subject = subject;

this.callbackHandler = callbackHandler;

}

@Override

public boolean login() throws LoginException {

NameCallback nameCB = new NameCallback("Username: ");

PasswordCallback passCB = new PasswordCallback("Password: ", false);

try {

callbackHandler.handle(new Callback[]{nameCB, passCB});

username = nameCB.getName();

String password = new String(passCB.getPassword());

// Hardcoded user check (replace with DB/LDAP lookup)

if ("admin".equals(username) && "admin123".equals(password)) {

principals.add(new UserPrincipal(username));

principals.add(new RolePrincipal("admin"));

return true;

}

} catch (Exception e) {

throw new LoginException("Login failed: " + e.getMessage());

}

throw new FailedLoginException("Invalid credentials");

}

@Override

public boolean commit() {

subject.getPrincipals().addAll(principals);

return true;

}

@Override

public boolean abort() {

return false;

}

@Override

public boolean logout() {

subject.getPrincipals().removeAll(principals);

return true;

}

}

**✅ 2. Principal Classes**

**UserPrincipal.java**

package com.example.auth;

import java.security.Principal;

public class UserPrincipal implements Principal {

private final String name;

public UserPrincipal(String name) { this.name = name; }

@Override public String getName() { return name; }

}

**RolePrincipal.java**

package com.example.auth;

import java.security.Principal;

public class RolePrincipal implements Principal {

private final String name;

public RolePrincipal(String name) { this.name = name; }

@Override public String getName() { return name; }

}

**✅ 3. jaas.config**

Create file: C:\tomcat\conf\jaas.config (or where Tomcat can access)

JAASWebApp {

com.example.auth.MyLoginModule required;

};

**✅ 4. Set Java Option**

Edit catalina.bat or catalina.sh to include:

-Djava.security.auth.login.config="C:\tomcat\conf\jaas.config"

Or set in setenv.bat:

set JAVA\_OPTS=-Djava.security.auth.login.config="C:\tomcat\conf\jaas.config"

**✅ 5. Configure server.xml (Tomcat Realm)**

Edit conf/server.xml inside <Engine>:

<Realm className="org.apache.catalina.realm.JAASRealm"

appName="JAASWebApp"

userClassNames="com.example.auth.UserPrincipal"

roleClassNames="com.example.auth.RolePrincipal"/>

**✅ 6. Web App Setup**

**web.xml inside JAASWebApp/WEB-INF**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">

<display-name>JAAS Web App</display-name>

<security-constraint>

<web-resource-collection>

<web-resource-name>Protected</web-resource-name>

<url-pattern>/secure/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

</auth-constraint>

<user-data-constraint>

<transport-guarantee>NONE</transport-guarantee>

</user-data-constraint>

</security-constraint>

<login-config>

<auth-method>FORM</auth-method>

<realm-name>JAASRealm</realm-name>

<form-login-config>

<form-login-page>/login.html</form-login-page>

<form-error-page>/error.html</form-error-page>

</form-login-config>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

</web-app>

**✅ 7. Sample Pages**

**login.html**

<form method="POST" action="j\_security\_check">

Username: <input type="text" name="j\_username" /><br/>

Password: <input type="password" name="j\_password" /><br/>

<input type="submit" value="Login" />

</form>

**secure/index.jsp**

<%@ page session="false" %>

<html>

<body>

<h2>Welcome: <%= request.getUserPrincipal().getName() %></h2>

<p>You are in the secure area.</p>

</body>

</html>

**error.html**

<h3>Login Failed. Try again.</h3>

**✅ 8. Package & Deploy**

* Compile the Java classes
* Package as WAR: JAASWebApp.war
* Deploy to tomcat/webapps/

**✅ Test**

Visit:

http://localhost:8080/JAASWebApp/secure/

Login:

* **Username:** admin
* **Password:** admin123

**✅ Summary**

| **Component** | **What It Does** |
| --- | --- |
| MyLoginModule | Custom JAAS login logic |
| jaas.config | Configures the login module |
| JAASRealm | Tomcat integration with JAAS |
| web.xml | Declares form-based login + roles |
| UserPrincipal | Carries user identity |
| RolePrincipal | Carries role name |