

141

Fundamentos básicos de
Segurança

Helder da Rocha
www.argonavis.com.br

- *Este capítulo apresenta*
 - *Noções básicas de segurança em J2EE*
 - *Exemplos de aplicações*
 - *Usos típicos de JAAS*
 - *Controle de acesso em aplicações J2EE com*
- *A abordagem deste assunto neste curso é **superficial**.*

Para maiores detalhes consulte

 - ***Livros-texto** (MEJB contém versão original do exemplo mostrado usando outro modelo de autenticação; J2EE Tutorial mostra exemplo usando container do J2EE-RI)*
 - ***Código-exemplo** (comentado)*
 - ***Capítulo 13 do curso J530** (www.argonavis.com.br)*
 - *Aplicações exemplo do capítulo 15 (código)*

Controle de acesso a autenticação em EJB

- A especificação EJB 2.0 define **controle de acesso** a métodos e **autenticação** de clientes
 - Não abrange outros aspectos importantes de segurança, como criptografia e comunicação segura
- Autenticação depende de implementação no container
 - Identidade é representada por objeto **java.security.Principal**
- O controle de acesso **declarado** no `ejb-jar.xml`
 - Declara-se papéis lógicos que podem ser mapeados a usuários e grupos: **<security-role>**
 - Papéis são associados a métodos para determinar quem tem autorização para executar: **<method-permission>**
 - Métodos podem ser executados em outros beans por um **Principal** diferente: **<security-identity>** **<run-as>**

Controle de acesso no ejb-jar.xml

```
<assembly-descriptor>
  <security-role>
    <role-name>BankCustomer</role-name>
  </security-role>
  <security-role>
    <role-name>BankAdmin</role-name>
  </security-role>
  <method-permission>
    <unchecked />
    <method>
      <ejb-name>CustomerEJB</ejb-name>
      <method-name>getPrimaryKey</method-name>
    </method>
  </method-permission>
  <method-permission>
    <role-name>BankAdmin</role-name>
    <method>
      <ejb-name>CustomerEJB</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
</assembly-descriptor>
```

Papéis lógicos participantes

ejb-jar.xml

Grupo de métodos não verificados

Grupo de métodos de CustomerEJB que só podem ser chamados por usuários que assumem o papel de BankAdmin

- *A autenticação depende de infraestrutura do servidor e pode ser implementada com o Java Authentication and Authorization Service - JAAS*
 - *JAAS é uma API para autenticação de usuários e autorização. A maior parte é implementada pelo servidor*
- *Principais classes*
 - *javax.security.auth.**Subject***
 - *javax.security.**Principal***
 - *javax.security.auth.callback.**Callback***
 - *javax.security.auth.callback.**CallbackHandler***
 - *javax.security.auth.login.**Configuration***
 - *javax.security.auth.login.**LoginContext***
 - *javax.security.auth.spi.**LoginModule***

Implementação do JAAS no JBoss (JBossSX)

- A implementação do JAAS no JBoss consiste de
 - *JAASecurityManager*
 - Módulos de configuração do servidor (*login-config.xml*) e do cliente (*auth.conf*)
 - Implementações de *javax.security.auth.spi.LoginModule*
- *ClientLoginModule*
 - Implementação de *LoginModule* no cliente
- *UsersRolesLoginModule*
 - Uma das implementações de *LoginModule* no servidor
 - Par de arquivos de texto para definir usuários e grupos
- *jboss.xml* e *jboss-web.xml*
 - Declaram domínio JAAS a ser usado através do elemento `<security-domain>` de *jboss.xml* e *jboss-web.xml*

Configuração do JBoss

- *Para utilizar os domínios de segurança do JBoss é preciso realizar configurações no servidor e em cada aplicação*
- *No servidor*
 - *Arquivo **login-config.xml** localizado no diretório `$JBOSS_HOME/server/default/conf/`*
 - *Criação de domínios de segurança e sua associação com módulos de login existentes*
- *Na aplicação*
 - *Arquivo `jboss.xml` (e/ou `jboss-web.xml`, se autenticação ocorrer via Web Container)*
 - *Declaração do domínio de segurança JAAS usado*
 - *Em clientes standalone, criação de domínios de segurança*

Domínio de segurança do servidor JBossSX

- Deve ser declarado em `jboss.xml` (ou `jboss-web.xml`)

```
<jboss>
  <security-domain>
    java:/jaas/dominio-servidor</security-domain>
  <session>
    <ejb-name>SecureHelloEJB</ejb-name>
    <jndi-name>login/HelloHome</jndi-name>
  </session>
</jboss>
```

`jboss.xml`

- Deve coincidir com nome de um `<application-policy>` (que define o domínio) em `login-config.xml`

```
<policy> (...) <application-policy name="dominio-servidor">
  <authentication>
    <login-module code="MyLoginModule" .../>
```

- Se não houver domínio com este nome em `login-config.xml`, será usado o domínio default "other", previamente configurado para usar `UsersRolesLoginModule` (veja slide seguinte)

Serviço JBossSX com UsersRolesLoginModule

- Módulo simples que utiliza par de arquivos de texto
 - É a configuração **default** em **login-config.xml**

```
<policy> (...)  
  <application-policy name="other">  
    <authentication>  
      <login-module  
        code="org.jboss.security.auth.spi.UsersRolesLoginModule"  
        flag="required" />  
    </authentication>  
  </application-policy>  
</policy>
```

Trecho de server/default/conf/login-config.xml

users.properties (nome=senha)

```
mickey=mouse  
niquel=nausea
```

Use preferencialmente
senha criptografada!

roles.properties (nome.Grupo=Role,...,Role)

```
mickey.Roles=TestRole, AdminRole  
niquel.Roles=NoRole, MouseRole
```

Coloque no CLASSPATH
do servidor (pode ser no EJB-JAR)

Domínio usando banco de dados

- Pode-se guardar nome, senha e grupo em banco de dados. Para isto, é preciso configurá-lo no JBoss (login-config.xml) com **DatabaseServerLoginModule**
 - É preciso também ter uma ou mais tabelas criadas com os dados de login (usuario, senha e grupo)
- O módulo requer três opções que são: **nome** do Datasource, **query** que retorne a **senha**, dado o **userid** e **query** que retorne um **grupo**, como "Roles" dado um **userid**

```
<application-policy name="dominio-JDBC-servidor">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule">
      <module-option name="dsJndiName">java:/DefaultDS</module-option>
      <module-option name="principalsQuery">
        select senha from usuarios where id=?</module-option>
      <module-option name="rolesQuery">
        select grupo, 'Roles' from grupos where id=?</module-option>
      <module-option name="unauthenticatedIdentity">nobody</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Domínio de segurança do cliente JBossSX

- Coloque no Classpath do cliente standalone
 - *Arquivo de configuração* (indica no mínimo a implementação de login module usada)

arquivo de configuração (default)

```
dominio-cliente {  
    // JBoss LoginModule implementation  
    org.jboss.security.ClientLoginModule required;  
};
```

- Informe ao cliente a localização do arquivo através da propriedade `java.security.auth.login.config`:

```
> java Prog -Djava.security.auth.login.config=auth.txt
```

- JARs do JBossSX: *jbosssx-client.jar* (além dos outros JARs necessários para o cliente JBoss)

Domínio de clientes dentro de containers

- *Para clientes que executam dentro do servidor (ex: servlets, JSP), chame o mesmo login module através de domínio definido na configuração do JBoss*
- *Ex: domínio declarado em login-config.xml:*

```
<application-policy name="dominio-cliente">
  <authentication>
    <login-module
      code="org.jboss.security.ClientLoginModule"
      flag="required">
    </login-module>
  </authentication>
</application-policy>
```

Cliente standalone padrão JAAS para login

```
public class HelloClient {
    public static void main(String[] args) throws Exception {
        LoginContext loginCtx = null;
        try {
            // Cria CallbackHandler
            CallbackHandler handler = new HelloCallbackHandler();
            // Carrega configuração
            loginCtx = new LoginContext("dominio-cliente", handler);
            // Faz o login
            loginCtx.login();
        } catch (LoginException e) {
            System.out.println("Login failed"); System.exit(1);
        }
        // Executa ação privilegiada propagando contexto de segurança
        Context ctx = new InitialContext(System.getProperties());
        Object obj = ctx.lookup("SecureHelloEJB");
        HelloHome home = (HelloHome)
            PortableRemoteObject.narrow(obj, HelloHome.class);
        Hello hello = home.create();
        System.out.println(hello.hello());
    }
}
```

Encapsula dados de autenticação

Faz o login aqui

Dominio do cliente (veja slides anteriores)

Chama métodos privilegiados

JAAS CallbackHandler

```
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;

public class HelloCallbackHandler implements CallbackHandler {
    private String username;
    private String password;

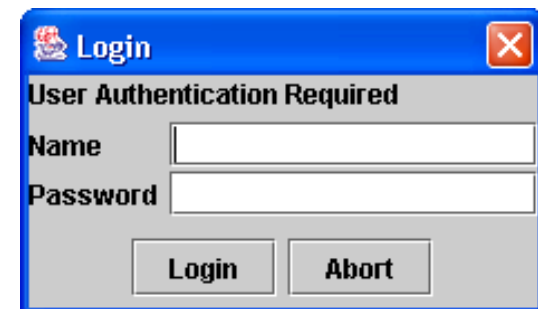
    public HelloCallbackHandler() {
        LoginDialog dialog = new LoginDialog("User Authentication Required");
        this.username = dialog.getData()[0];
        this.password = dialog.getData()[1];
    }
    public void handle(Callback[] callbacks)
        throws java.io.IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof NameCallback) {
                NameCallback nc = (NameCallback) callbacks[i];
                nc.setName(username);
            } else if (callbacks[i] instanceof PasswordCallback) {
                PasswordCallback pc = (PasswordCallback) callbacks[i];
                pc.setPassword( this.preparePassword(password) );
            } else throw new UnsupportedCallbackException(callbacks[i], "Error");
        }
    }
    private char[] preparePassword(String word) {
        return word.toCharArray(); // ou rotina de criptografia
    }
}
```

GUI simples para entrada de texto

Preenche array de callbacks recebido com nome e senha lidos

Exemplo JAAS: como executar

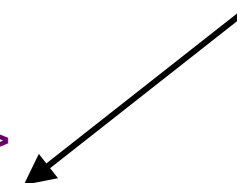
- *Diretório cap15/exemplos/mejb*
- *Configuração*
 - *Configure build.properties*
 - *Edite usuários e grupos em lib/users.properties e lib/roles.properties*
- *Use targets do Ant*
 - > *ant jboss.deploy* (para instalar)
 - > *ant run.jboss.client* (para rodar)
- *Digite nome e senha compatíveis com arquivos de configuração*
 - *Tente logar como usuário válido não autorizado (niquel)*



Aplicação exemplo: ejb-jar.xml

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>SecureHelloEJB</ejb-name>
      <home>examples.HelloHome</home>
      <remote>examples.Hello</remote>
      <ejb-class>examples.HelloBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>TestRole</role-name>
    </security-role>
    <method-permission>
      <role-name>TestRole</role-name>
      <method>
        <ejb-name>SecureHelloEJB</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```

*Todos os métodos
acessíveis a TestRole*



Veja código exemplo em cap 15/exemplos/mejb

Controle de acesso e autenticação em Servlets

- A especificação Servlet 2.3 permite a configuração de segurança em dois domínios
 - **Autenticação**: o processo de verificação da identidade do usuário que solicita um recurso - quem é?
 - **Autorização**: processo de verificação das permissões que um usuário autenticado possui - o que ele pode fazer?
- Os dois recursos podem ser configurados para cada contexto no web.xml definindo
 - **Login Configuration**: configuração de métodos de autenticação utilizados
 - **Security Roles**: perfis de usuário para autorização
 - **Security Constraint**: URLs e métodos de acesso permitidos e perfis de segurança necessários para acessá-los

Configuração de Login em aplicações Web

- Escolha uma dentre quatro técnicas de autenticação
 - **BASIC**: mostra uma janela do browser que recebe nome e senha. Os dados são enviados em conexão insegura
 - **FORM**: igual a BASIC mas em vez de usar uma janela do browser, permite o uso de um formulário HTML
 - **DIGEST**: usa criptografia fraca para enviar os dados
 - **CLIENT-CERT**: requer certificado X-509 para funcionar e usa criptografia forte (128-bit)
- **BASIC, FORM e DIGEST** são seguros se usados com SSL

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/form.html</form-login-page>
    <form-error-page>/erro.html</form-error-page>
  </form-login-config>
</login-config>
```

```
<login-config>
  <auth-method>
    BASIC
  </auth-method>
</login-config>
```

Autorização: perfis de usuário - web.xml

- Uma vez definida a forma de autenticação, é preciso definir *perfis de usuário* habilitados a acessar os recursos
- A *declaração* de perfis de usuários é feita no web.xml usando `<security-role>`. A associação desses perfis com usuários reais é dependente de servidor

```
<security-role>
  <role-name>administrador</role-name>
</security-role>

<security-role>
  <role-name>membro</role-name>
</security-role>

<security-role>
  <role-name>visitante</role-name>
</security-role>
```

Web-Resource Collection no web.xml

- A coleção de recursos Web protegidos e os métodos de acesso que podem ser usados para acessá-los é definido em um bloco `<web-resource-collection>` definido dentro de `<security-constraint>`
 - Inclui URL-patterns `<url-pattern>` que indicam quais os mapeamentos que abrangem a coleção
 - Inclui um `<http-method>` para cada método permitido

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Seção de Assinantes</web-resource-name>
    <url-pattern>/membros/*</url-pattern>
    <url-pattern>/preferencias/config.jsp</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  ...
</security-constraint>
```

Security Constraint no web.xml

- O **Web Resource Collection** faz parte do **Security Constraint** que associa perfis de usuário (papéis lógicos) à coleção

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Seção de Assinantes</web-resource-name>
    <url-pattern>/membros/*</url-pattern>
    <url-pattern>/preferencias/config.jsp</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>

  <auth-constraint>
    <role-name>administrador</role-name>
  </auth-constraint>
</security-constraint>
...
<security-constraint>
  ... outras coleções e constraint ...
</security-constraint>
```

Form-based Login no web.xml

- Para autenticação por formulário, é preciso definir no `<login-config>` **FORM** como `<auth-method>`
- Adicionalmente, é informada a página que implementa o formulário. Esta página deve conter um elemento `<FORM>` HTML com algumas restrições
 - Atributo `ACTION` de `<FORM>` deve conter: **`j_security_check`**
 - Campo `<INPUT>` do nome deve conter: **`j_username`**
 - Campo `<INPUT>` da senha deve conter: **`j_password`**

```
<form action="j_security_check" method="POST">  
  <p>Digite seu nome de usuário:  
  <input type="text" name="j_username">  
  <p>Digite sua senha:  
  <input type="password" name="j_password">  
  <input type="submit">  
</form>
```

Como identificar o principal

- O *principal* (a pessoa que foi autenticada) é acessível dentro de EJBs e servlets/JSP através de seus contextos

- Ambos retornam *javax.security.Principal*
- *getName()* de *Principal* retorna id do usuário logado

- *EJBContext: getCallerPrincipal()*

```
Principal usuario = ctx.getCallerPrincipal();  
String nome = usuario.getName();
```

- *ServletContext: getUserPrincipal()*

```
Principal usuario =  
    application.getUserPrincipal();  
String nome = usuario.getName();
```

Como identificar o papel

- O papel (role) do usuário logado pode ser testado também através dos contextos de EJBs e Servlets usando métodos similares

- Nome do papel pode deve ser mapeado a outro nome (referência) no deployment descriptor

- *EJBContext: boolean isCallerInRole(String role)*

```
if (ctx.getCallerInRole("admin")) {  
    // dar acesso ao usuário  
}
```

- *ServletContext: boolean isUserInRole(String role)*

```
if (application.getUserInRole("admin")) {  
    // dar acesso ao usuário  
}
```


Configuração do JBoss

- *Para executar os exercícios e os outros exemplos (deste e dos próximos capítulos) é preciso configurar a tabela com usuários e estabelecer um domínio de segurança no JBoss*
- *Roteiro*
 - *Abra o arquivo **login-config.xml** e inclua as duas definições de domínio contidas no arquivo **cap14/jaas.xml**. Ele define um LoginModule para acesso JDBC.*
 - *Rode **ant create-table** para criar as tabelas e povoá-las com os usuários e grupos necessários*
 - *Reinicie o JBoss para que ele leia as configurações*

- *1. a) Crie uma aplicação contendo um session bean com dois métodos*
 - *listarNomes()*
 - *adicionarNome(String nome)*
- *1. b) Crie dois papéis (roles): **User** e **Admin** e defina as permissões dos métodos*
 - *Admin: pode listarNomes() e adicionarNome()*
 - *User: pode listarNomes()*
- *1. c) Implemente um cliente que faça o login e chame um dos dois métodos*
- *1. d) Configure usuários do sistema, associe-os aos papéis existentes e rode a aplicação*

- [1] *JBoss Group. JBoss User's Manual. Chapter 8: The JBoss Security Extension Framework.*
- [2] *Erik Jendrock. Security. Sun J2EE Tutorial.*
- [3] *Richard Monson-Haefel, Enterprise JavaBeans, 3rd. Edition, O'Reilly and Associates, 2001*
- [4] *Ed Roman. Mastering EJB 2.*

helder@argonavis.com.br

argonavis.com.br