



Open-source Education →

Segurança declarativa J2EE

Iniciativa Globalcode

Agenda

1. Conceitos sobre segurança
2. Segurança em aplicativos e o modelo J2EE
3. Protegendo o Web Container
4. Protegendo o EJB Container
5. Acesso ao Container protegido com JAAS
6. Conclusões

Palestrante

Vinicius Senger – vinicius@globalcode.com.br

- Sócio e fundador da Globalcode, foi instrutor e consultor da Sun e Oracle no Brasil;
- Trabalhou em projetos de grande porte em bancos. Começou a programar com 8 anos e trabalha com desenvolvimento de softwares profissionalmente desde os 13 anos;
- Certificações: Sun Certified Java Programmer e Sun Enterprise Architect – p1;

Agenda

1. Conceitos sobre segurança
2. Segurança no modelo J2EE
3. Protegendo o Web Container
4. Protegendo o EJB Container
5. Acesso ao Container protegido com JAAS
6. Conclusões

Conceitos sobre segurança

- ARPANET em 1969 não tinha preocupações com segurança;
- Redes ARPANET tinham como principal objetivo a inteligência de rotas e não nos dados;
- Primeiro “ataque” ARPANET aconteceu em 1988: “**The Morris Worm**”;
- O foco da Internet mudou completamente, tanto em segurança, quanto em aplicação;

Conceitos sobre segurança

- Incidente de segurança: qualquer violação de política de acesso;
- Probe (sondar / investigar): forma para obter uma conta de acesso ou tentar um acesso;
- Scan (varredura): forma de descobrir quais portas oferecem serviços;
- Comprometimento de conta: uso de uma conta roubada / adquirida ilegalmente;

Conceitos sobre segurança

- Comprometimento do root: pior caso;
- Sniffer (cheirador): captura todos os pacotes de uma determinada rede;
- Denial of service (negação de serviço): tipicamente acontece através de “bombardeios” no servidor;
- Código malicioso: algum programa ilegal é transferido e executado;
- Alguns fatos sobre TCP/IP Vs. Guerras;

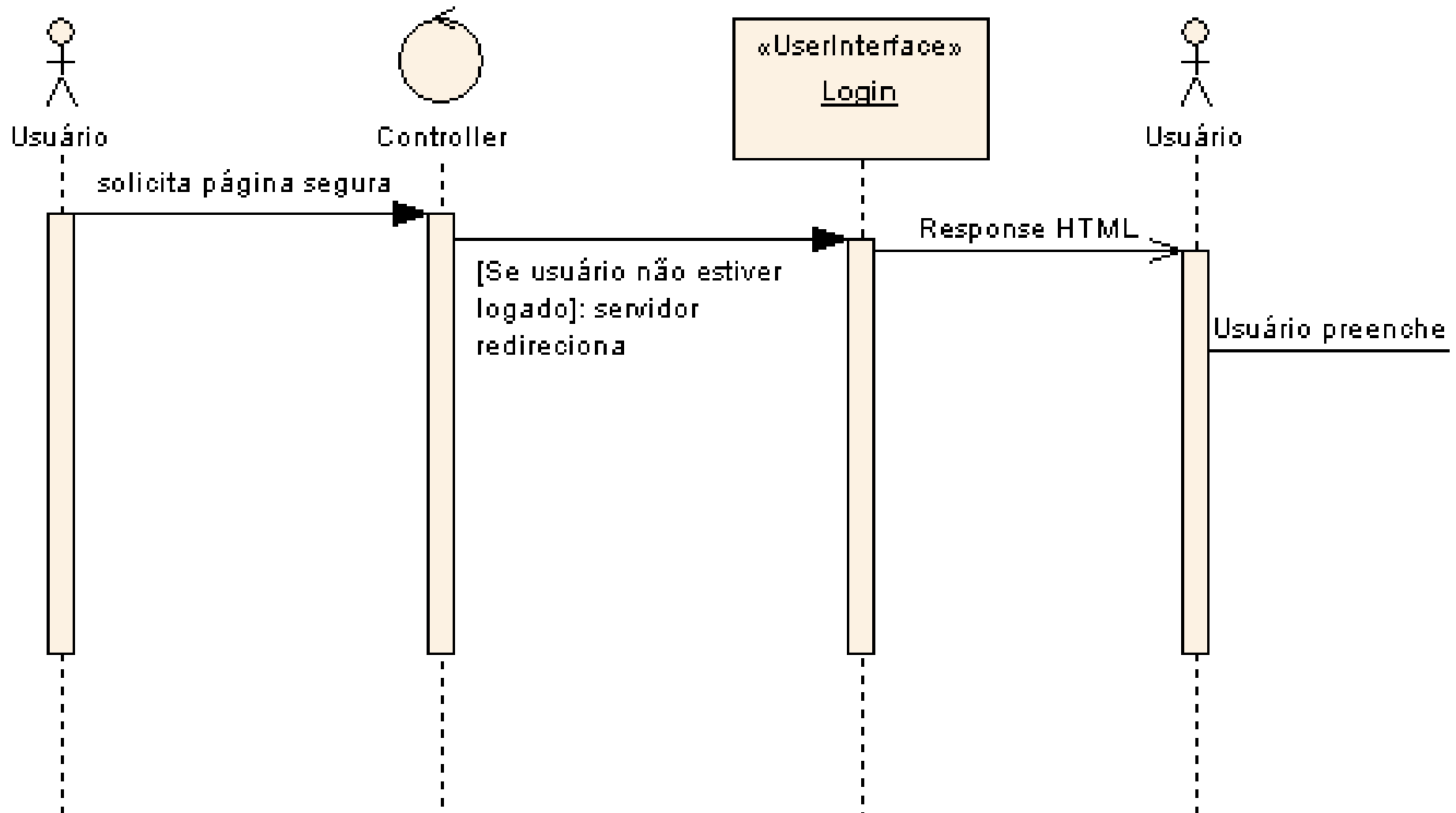
Agenda

1. Conceitos sobre segurança
2. Segurança em aplicativos e o modelo J2EE
3. Protegendo o Web Container
4. Protegendo o EJB Container
5. Acesso ao Container protegido com JAAS
6. Conclusões

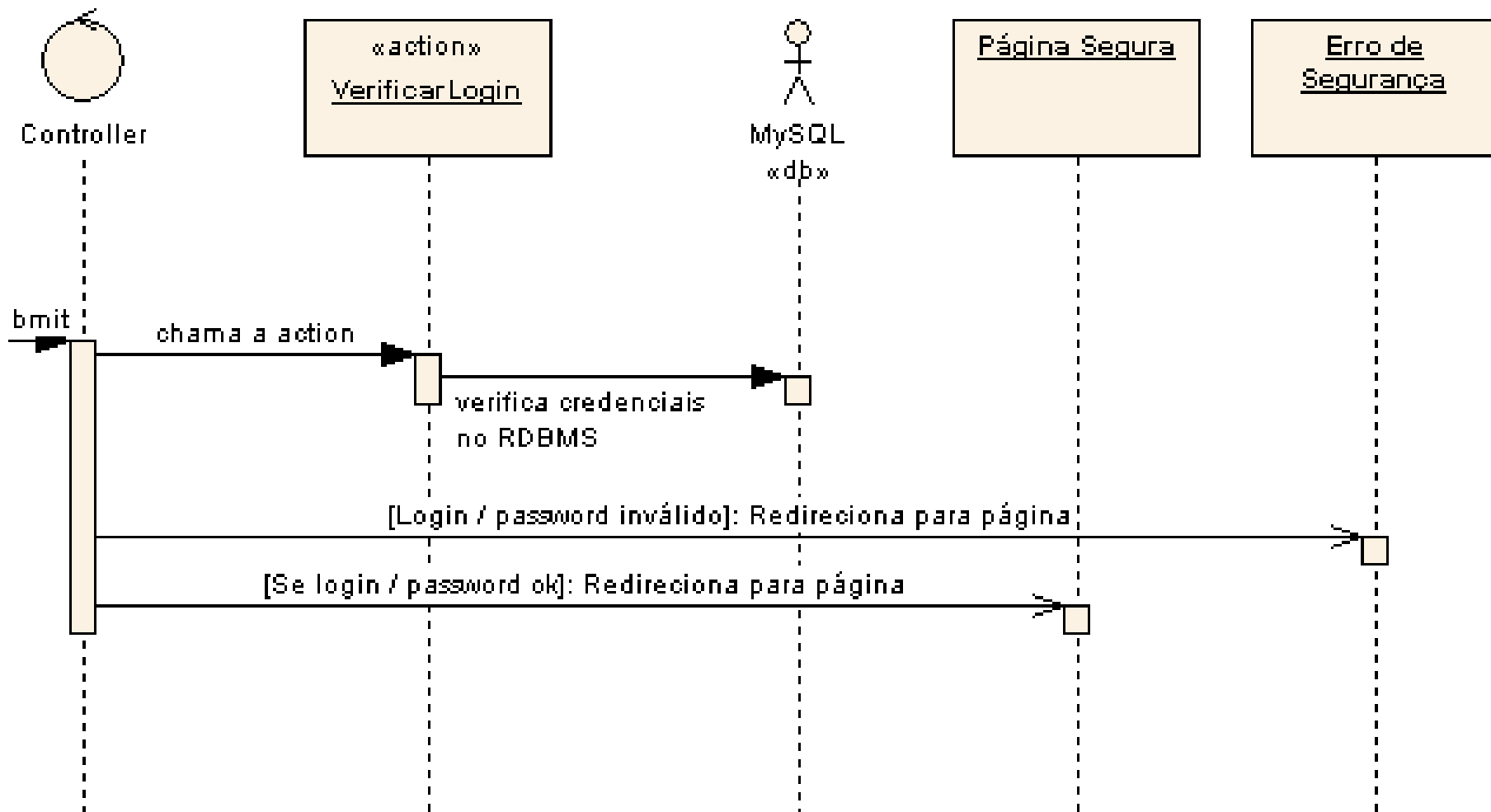
Segurança em aplicativos

- Ao desenvolver um aplicativo “seguro”, devemos considerar:
 1. Criação / Configuração de um repositório de usuários;
 2. Tela para capturar login / senha usuário;
 3. Funcionalidade que efetua a validação do login / senha no repositório;
 4. Funcionalidade que verifica se usuário tem permissão de acesso ao recurso solicitado;
 5. Em caso de falta de permissão ou login / senha inválido, envia mensagem / exception;

Segurança em aplicativos



Segurança em aplicativos



Segurança em aplicativos

Tarefa	Responsável
Planejamento de regras e identificação de papéis.	Analista de Sistemas.
Modelagem de estratégias técnicas.	Arquiteto J2EE.
Criação de repositório de usuários.	Administrador de rede / segurança.
Integração do container J2EE com o repositório.	Administrador de servidores J2EE.
Desenvolver EJB com tags de segurança.	Desenvolvedor EJB's.
Implementação do EJB.	Administrador de servidores J2EE.
Acesso ao EJB com aplicativos client Swing ou Web.	Desenvolvedor Java.

Segurança em aplicativos

- Podemos utilizar os seguintes níveis de segurança:
 - 1. Sem segurança:** sistema totalmente aberto, não requer login válido;
 - 2. Nível básico:** o usuário precisa informar login e senha, porém o sistema não tem criptografia;
 - 3. Nível intermediário:** as informações login e senha são criptografadas, porém o restante não;
 - 4. Nível avançado:** tudo é criptografado com chaves;
 - 5. Nível máximo:** tudo é criptografado e o usuário “prova” sua identidade através de um certificado digital;

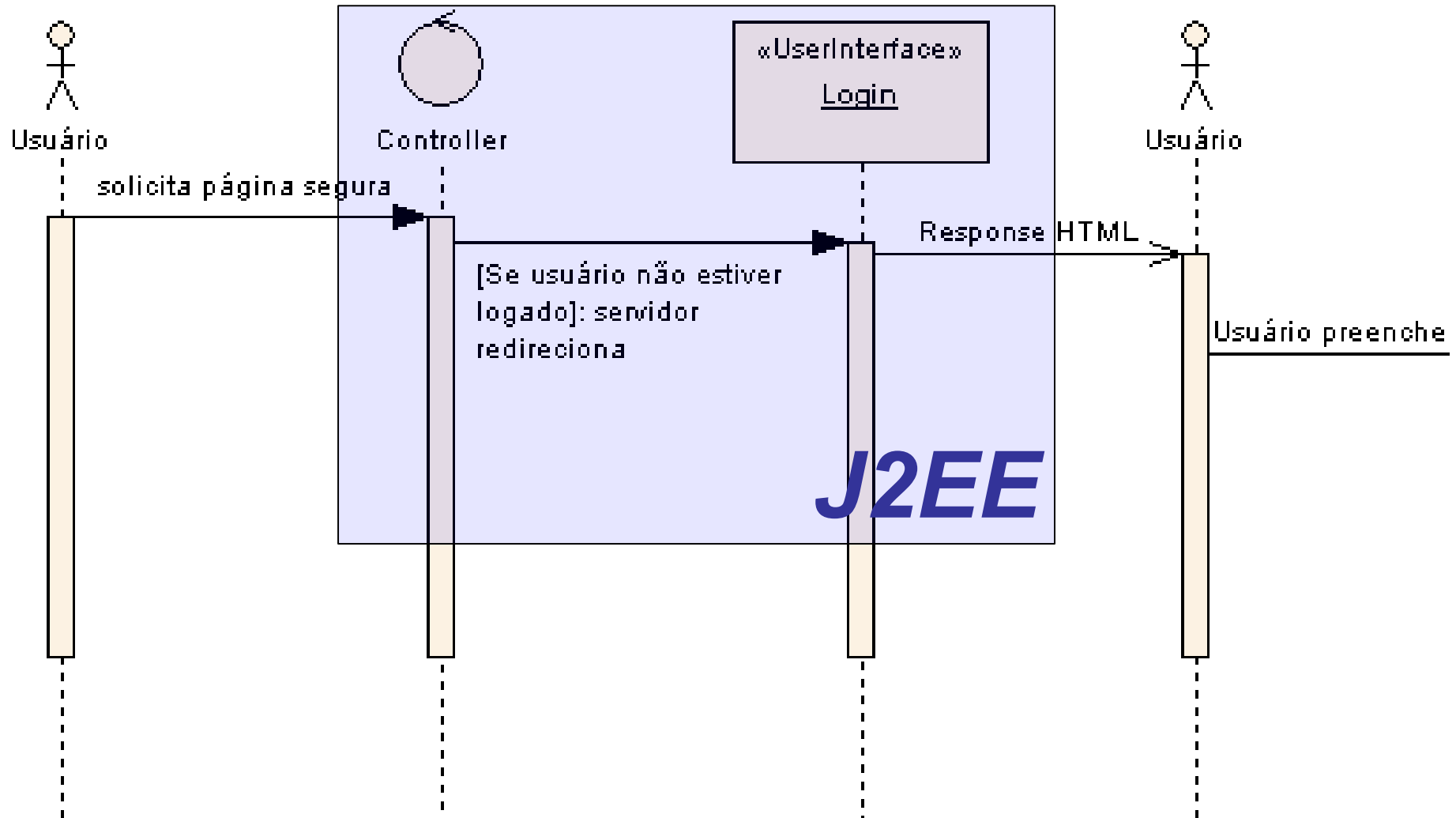
Segurança no J2EE

- **J2EE suportada todos** os níveis e permite o desenvolvimento de aplicativos seguros através de:
 - **Segurança programada**: o código é totalmente responsável por toda a segurança;
 - **Segurança declarada**: delegamos o controle de segurança para o container;
 - **Segurança mista**: declaramos o básico, programamos o avançado;

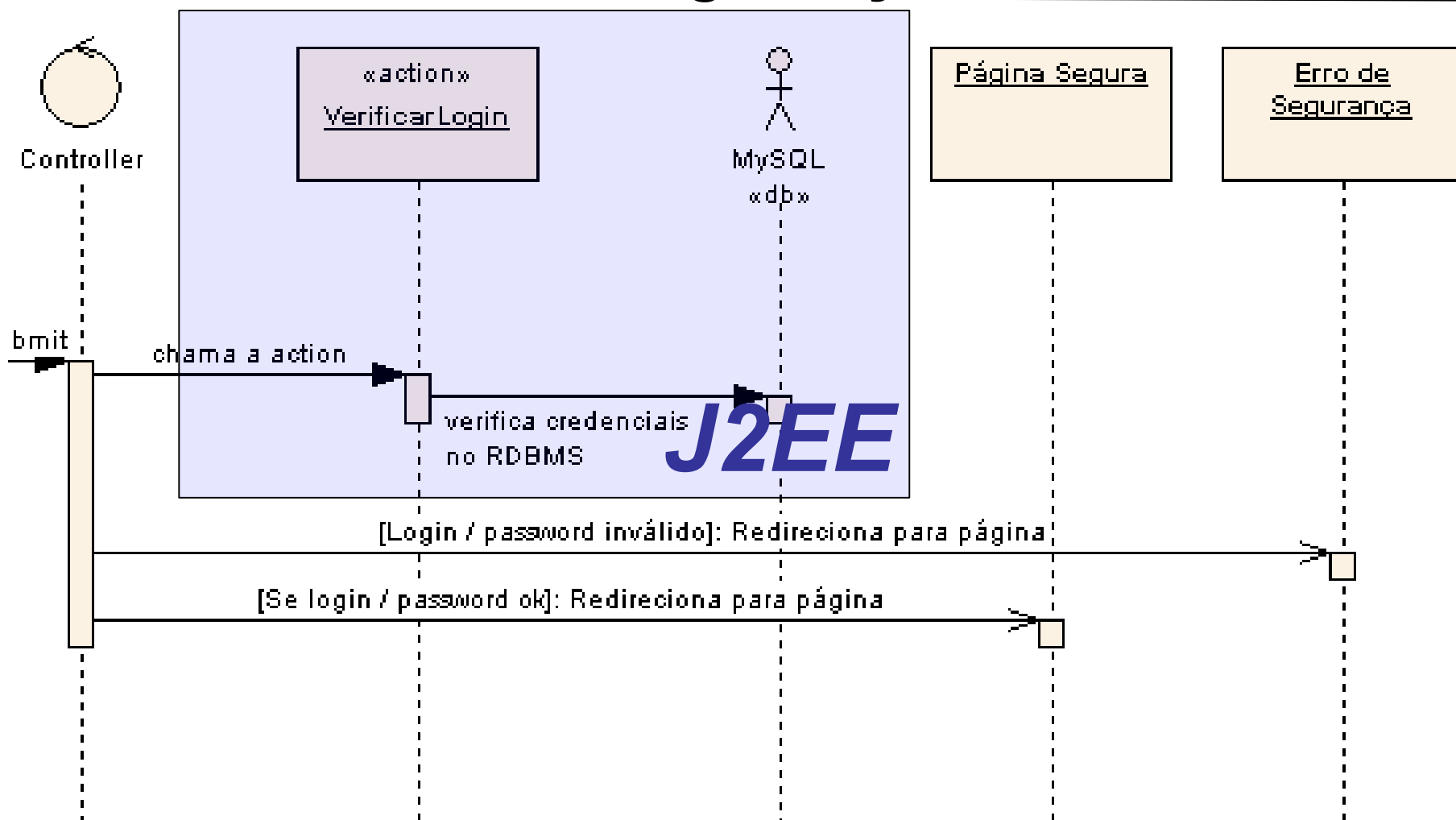
Segurança no J2EE

- Das necessidades de um sistema de segurança, o J2EE oferece:
 1. **Comunicação** com seu repositório de usuários (RDBMS, arquivos, LDAP, NIS etc.);
 2. **Verificar login** / password no repositório de usuários;
 3. Checar a **segurança** em cada **acesso** a EJB's, Servlets, JSP, etc.;
 4. **Integrar segurança** do container **Web** com o container de **EJBs**;

Segurança no J2EE



Segurança no J2EE



Como?

- Para utilizarmos o serviço de segurança da plataforma J2EE devemos:
 1. Configurar um repositório de usuários no application server (JBoss, Tomcat, Websphere, WebLogic, etc.);
 2. Declarar características de segurança no web.xml e / ou ejb-jar.xml;
 3. Vincular o aplicativo ao repositório de usuário configurado no A.S. através do deployment descriptor específico;

Segurança mista

- Para regras avançadas devemos misturar segurança declarativa com segurança programa;
- Ex. Usuários do grupo “gestão contas” podem acessar recurso X das 9:00 as 18:00;
- Dentro de componentes Web e EJB podemos acessar métodos que indicam o grupo do usuário / usuário em questão;

Agenda

1. Conceitos sobre segurança
2. Segurança em aplicativos e o modelo J2EE
3. Protegendo o Web Container
4. Protegendo o EJB Container
5. Acesso ao Container protegido com JAAS
6. Conclusões

Web Container

- As seguintes tarefas devem ser executadas:
 1. Escolha de um repositório de usuários (RDBMS, XML, NIS, LDAP, ...) e cadastramento dos usuários / grupos no repositório;
 2. Configuração de repositório de usuários (REALM) no seu application server;
 3. Definição de grupos (roles) e recursos (jsp, servlet, ...) associados aos grupos (web.xml);
 4. Vínculo do aplicativo WAR com o repositório (ex. jboss-web.xml);

Web Container - Demo

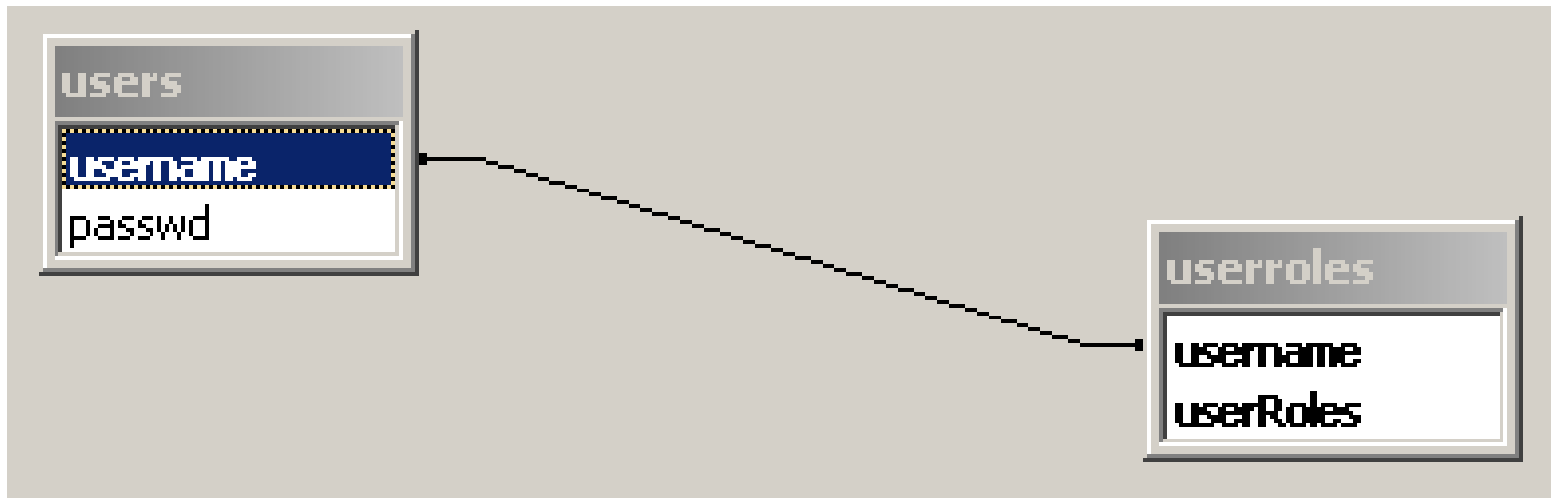
- Vamos analisar um aplicativo Web com segurança declarada no web.xml, nesta demonstração vamos:
 1. Utilizar como repositório de usuários, o banco de dados MySQL;
 3. Utilizar JBoss;

Web Container - Demo

1► Repositório MySQL com duas tabelas:

user – contém usuários e senhas

userroles – contém grupos que o usuário pertence



Web Container - Demo

1► Repositório MySQL com duas tabelas:

```
CREATE TABLE `userroles` (  
  `username` varchar(64) default NULL,  
  `userRoles` varchar(64) default NULL  
) TYPE=MyISAM;
```

```
CREATE TABLE `users` (  
  `username` varchar(64) NOT NULL default "",  
  `passwd` varchar(64) default NULL,  
  PRIMARY KEY (`username`)  
) TYPE=MyISAM;
```


Web Container - Demo

2► Configurar o repositório MySQL no Jboss

2.1 Abra o arquivo login-config.xml no diretório conf do jboss

2.2 Na tag <policy> adicione:

```
<policy>
```

```
  <application-policy name = "s1">
```

```
    <authentication>
```

```
      <login-module
```

```
        code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
```

```
        flag="required">
```

Web Container - Demo

2.3 Indicamos datasource e comandos SQL

```
<module-option name="dsJndiName">
```

```
java:/AAPooling1
```

```
</module-option>
```

```
<module-option name="principalsQuery">
```

```
select passwd from users where username=?
```

```
</module-option>
```

```
<module-option name="rolesQuery">
```

```
select userRoles from userroles where username=?
```

```
</module-option>
```

Web Container - Demo

2.4 Fechamos as tags

</login-module>

</authentication>

</application-policy>

Web Container - Demo

2► Resultado final

```
- <policy>
- <application-policy name="s1">
- <authentication>
- <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
  flag="required">
  <module-option name="dsJndiName">java:/AAPooling1</module-option>
  <module-option name="principalsQuery">select passwd from users username where
    username=?</module-option>
  <module-option name="rolesQuery">select userRoles, 'Roles' from userroles where
    username=?</module-option>
  </login-module>
</authentication>
</application-policy>
```

Web Container - Demo

3► Definição de segurança do aplicativo (war)

3.1 Abra o arquivo web.xml e configure as seguintes tags e configure a forma de login:

```
<login-config>  
  <auth-method>BASIC</auth-method>  
  <realm-name>Mini-curso Globalcode</realm-name>  
</login-config>
```

Também podemos configurar DIGEST ou FORM;

Web Container - Demo

3.2 Configure os grupos / roles do aplicativo:

```
<security-role>
  <role-name>basic-user</role-name>
</security-role>
<security-role>
  <role-name>manager-user</role-name>
</security-role>
<security-role>
  <role-name>root-user</role-name>
</security-role>
```

Web Container - Demo

3.3 Configure os recursos protegidos e também quais grupos acessam:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      JSPs do aplicativo
    </web-resource-name>
    <url-pattern>*.jsp</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>basic-user</role-name>
  </auth-constraint>
</security-constraint>
```

Web Container - Demo

3► Resultado final 1/3

```
<web-app>
  <context-param>
    <param-name>lab</param-name>
    <param-value>Laboratorio 2</param-value>
  </context-param>
  <resource-ref>
    <res-ref-name>jdbc/AAPooling1</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
```


Web Container - Demo

3► Resultado final 2/3

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      JSPs do aplicativo
    </web-resource-name>
    <url-pattern>*.jsp</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>basic-user</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Mini-curso Globalcode</realm-name>
</login-config>
```

Web Container - Demo

3► Resultado final 3/3

```
<security-role>
  <role-name>basic-user</role-name>
</security-role>
<security-role>
  <role-name>manager-user</role-name>
</security-role>
<security-role>
  <role-name>root-user</role-name>
</security-role>
</web-app>
```

Web Container - Demo

3► Observações:

3. Requisitos de criptografia também podem ser configurados com a tag <transport-guarantee>;
4. Você pode restringir o acesso somente ao método POST e / ou GET;
3. A melhor maneira para você aprender é através do deploytool;

Web Container - Demo

3 ► Observações - Deploytool

Files.Applications.testes de client.WebApp1

Filter Mapping JNDI Names Resource Env. Refs Resource Refs Security Roles
General Context Env. Entries EJB Refs Event Listeners File Refs

User Authentication Method:
Form Based Settings...

Security Constraints

SecurityConstraint	Add	Delete...
SecurityConstraint		

Web Resource Collections

*.do	Add	Delete...
WRCollection		

Network Security Requirement:
CONFIDENTIAL

Authorized Roles

teste

☒ GET
☒ POST
Edit...

Edit...

Web Container - Demo

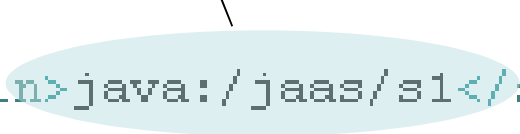
- 3 ► Observações - conhecendo usuário / grupo no código:
3. Podemos conhecer o usuário no servlet através de **request.getUserPrincipal();**
 4. Podemos perguntar seu usuário requisitante pertence a uma rola através do método **request.isUserInRole("root-user");**

Web Container - Demo

4► Vinculamos o aplicativo com o repositório

4.1 Abra o arquivo jboss-web.xml e adicione a seguinte tag:

Configuramos no login-config.xml, etapa 2



```
<jboss-web>
  <security-domain>java:/jaas/s1</security-domain>
  <resource-ref>
    <res-ref-name>jdbc/AAPooling1</res-ref-name>
    <jndi-name>java:/jdbc/GlobalcodeDS</jndi-name>
  </resource-ref>
</jboss-web>
```

Web Container - Demo

RESULTADO FINAL

3. Ao acessar qualquer página JSP, o JBoss verifica se o usuário está logado;
4. Se não estiver, envia janela de login, verifica login / password no MySQL;
5. Obtendo sucesso no login, o JBoss verifica se o usuário pertence ao grupo que tem acesso ao recurso;

Agenda

1. Conceitos sobre segurança
2. Segurança em aplicativos e o modelo J2EE
3. Protegendo o Web Container
- 4. Protegendo o EJB Container**
5. Acesso ao Container protegido com JAAS
6. Conclusões

Protegendo EJB's

- Um **EJB** sem configurações de **segurança** representa uma **grande vulnerabilidade** na rede interna da **empresa**;
- O serviço **JNDI** **difícilmente** é restringido por **firewall**;
- Você **já paga** o custo de segurança de EJB;
- A autenticação no **Web Container** pode ser **propagada** para o **EJB Container**;

Protegendo EJB's

- As seguintes tarefas devem ser executadas:
 1. *Escolha de um repositório de usuários (RDBMS, XML, NIS, LDAP, ...) e cadastramento dos usuários / grupos no repositório (idem web)*
 2. *Configuração de repositório de usuários (REALM) no seu application server (idem web)*
 3. Definição de grupos (roles) e recursos (ejb's) associados aos grupos (ejb-jar.xml);
 4. Vínculo do componente com o repositório (ex. jboss.xml);

Protegendo EJB's - DEMO

1 & 2 ► Repositório e configuração de usuários / grupos

Vamos utilizar as mesmas configurações apresentadas na web:

- MySQL como repositório;
- login-config.xml com nome de S1;

Protegendo EJB's - DEMO

3► Definição de segurança no componente (ejb-jar)

3.1 Abra o arquivo ejb-jar.xml e adicione as seguintes tags:

```
<security-identity>  
    <description />  
    <use-caller-identity />  
</security-identity>
```

Esta tag indica: utilize as credenciais do solicitante, poderíamos fixar uma role para execução.

Protegendo EJB's - DEMO

3.2 Defina os grupos / roles que serão posteriormente associados aos métodos dos EJB's:

```
<assembly-descriptor>
  <security-role>
    <description />
    <role-name>curioso</role-name>
  </security-role>
  <security-role>
    <description />
    <role-name>basic-user</role-name>
  </security-role>
```

Protegendo EJB's - DEMO

3.3 Associe os métodos aos grupos / roles:

```
<method-permission>
  <role-name>curioso</role-name>
  <method>
    <description />
    <ejb-name>AnoBissexto</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
</assembly-descriptor>
```

Protegendero EJB's - DEMO

3► Resultado final 1/3

```
<ejb-jar>
  <description>Academia do Arquiteto</description>
  <display-name>Exemplo01</display-name>
  <enterprise-beans>
    <session>
      <display-name>AnoBissexto</display-name>
      <ejb-name>AnoBissexto</ejb-name>
      <home>
        br.com.globalcode.aa.ejb.session.AnoBissextoHome
      </home>
      <remote>
        br.com.globalcode.aa.ejb.session.AnoBissexto
      </remote>
      <ejb-class>
        br.com.globalcode.aa.ejb.session.AnoBissextoBean
      </ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
```

Protegendo EJB's - DEMO

3► Resultado final 2/3

```
<security-identity>  
  <description />  
  <use-caller-identity />  
</security-identity>  
  
</enterprise-beans>
```


Protegendero EJB's - DEMO

3► Resultado final 3/3

```
<assembly-descriptor>
  <security-role>
    <description />
    <role-name>curioso</role-name>
  </security-role>
  <security-role>
    <description />
    <role-name>basic-user</role-name>
  </security-role>
  <method-permission>
    <role-name>curioso</role-name>
    <method>
      <description />
      <ejb-name>AnoBissexta</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
</assembly-descriptor>
<ejb-client-jar />
</ejb-jar>
```

Protegendo EJB's - DEMO

4► Vinculamos o componente com o repositório

4.1 Abra o arquivo jboss.xml e adicione a seguinte tag:

```
<jboss>
  <security-domain>java:/jaas/s1</security-domain>
  <enterprise-beans>
    <session>
      <ejb-name>AnoBissexto</ejb-name>
      <jndi-name>AnoBissexto</jndi-name>
      <!-- <clustered>true</clustered> -->
    </session>
  </enterprise-beans>
</jboss>
```

Protegendo EJB's - DEMO

RESULTADO FINAL

- “Agora você precisa ser alguém para acessar o EJB”;
- Se utilizar Web, basta autenticar na Web;
- Se utilizar cliente stand-alone, precisaremos do JAAS;

DEMO: Tomando um NãoPodeException...

Agenda

1. Conceitos sobre segurança
2. Segurança em aplicativos e o modelo J2EE
3. Protegendo o Web Container
4. Protegendo o EJB Container
5. Acesso ao Container protegido com JAAS
6. Conclusões

JAAS

- Java Authentication and Authorization Service;
- API padronizada para autenticação e autorização;
- Mecanismo de “plugabilidade” torna a API capaz de trabalhar com qualquer repositório;
- Pode trabalhar com Windows, Unix, Kerberos / LDAP, JNDI...
- Neste mini-curso, vamos utilizar para JNDI, pois queremos acessar um EJB seguro;

JAAS - DEMO

- ▶ Para acessar um EJB seguro, precisamos configurar / desenvolver os seguintes artefatos JAAS:
 1. Desenvolver uma implementação da interface **javax.security.auth.callback.CallbackHandler**
 2. Criar arquivo de configuração JAAS
 3. Criar um objeto da sua classe “CallbackHandler” antes do lookup JNDI
 4. Na execução da classe configurar
 - Djava.security.auth.login.config==/jaas.config

JAAS - DEMO

1► Implementação de CallbackHandler

1.1 Devemos escrever uma classe de interpretação da solicitação de login do container:

```
package br.com.globalcode.aa.swing;

import java.io.*;

public class LoginCallbackHandle implements CallbackHandler {
    private String login;
    private String senha;
    public LoginCallbackHandle(String login, String senha) {
        this.login = login;
        this.senha = senha;
    }
}
```

JAAS - DEMO

```
public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException {
    for (int i = 0; i < callbacks.length; i++) {
        if (callbacks[i] instanceof TextOutputCallback) {
            TextOutputCallback toc = (TextOutputCallback) callbacks[i];
            System.out.println(toc.getMessage());
        } else if (callbacks[i] instanceof NameCallback) {
            NameCallback nc = (NameCallback) callbacks[i];
            nc.setName(login);
        } else if (callbacks[i] instanceof PasswordCallback) {
            PasswordCallback pc = (PasswordCallback) callbacks[i];
            pc.setPassword(senha.toCharArray());
        } else {
            throw new UnsupportedCallbackException(callbacks[i],
                "Callback JAAS não suportado pelo aplicativo cliente.");
        }
    }
}
```


JAAS

- JAAS assusta a primeira vista...



JAAS - DEMO

2► Arquivo de configuração JAAS:

2.1 Configurar o provider / módulo de autenticação:

```
other {  
    org.jboss.security.ClientLoginModule required  
    ;  
};
```

JAAS - DEMO

3► Criar objeto CallHandler antes de chamar o EJB

3.1 Criamos o objeto e informamos login e senha:

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        try {  
            String login = args.length > 0 ? args[0] : "vsenger";  
            String senha = args.length > 0 ? args[1] : "senha";  
            String anoString = args.length > 0 ? args[2] : "2001";  
            LoginContext lc = null;  
            lc = new LoginContext("other",  
                                new LoginCallbackHandle(login, senha));  
            lc.login();  
        }  
    }  
}
```

JAAS - DEMO

3.1 Criamos o objeto e informamos login e senha:

```
lc.login();  
Context context = ServiceLocator.getInitialContext();  
Object objref = context.lookup("java:/AnoBissexto");  
AnoBissexto ab = null;  
AnoBissextoHome anoHome = null;  
anoHome = (AnoBissextoHome) PortableRemoteObject  
    .narrow(objref, AnoBissextoHome.class);  
ab = anoHome.create();  
int ano = Integer.parseInt(anoString);  
boolean anobissexto = ab.anoBissexto(ano);  
System.out.println("Ano "  
    + (anobissexto ? "é" : "não é") + " bissexto.");
```

JAAS - DEMO

RESULTADO FINAL

3. Criamos um objeto JAAS, do tipo CallbackHandler;
4. Instanciamos e chamamos o método login();
5. Ao acessar o EJB, a API JNDI automaticamente utilizará este objeto;



Rodando...

Agenda

1. Conceitos sobre segurança
2. Segurança em aplicativos e o modelo J2EE
3. Protegendo o Web Container
4. Protegendo o EJB Container
5. Acesso ao Container protegido com JAAS
6. Conclusões

***O seguro morreu porque você não
configurou o documento XML***



Open-source Education →

Segurança declarativa J2EE

Iniciativa Globalcode