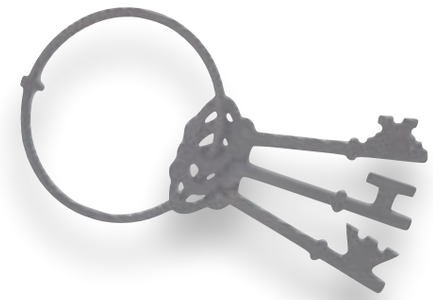
THE DEVELOPER'S CONFERENCE

TRILHA JAVA EE

florianópolis
abril 2017



A API de Segurança do **Java EE 8** (JSR-375)

summa
Technology + Business

helder da rocha

Objetivos

- Como era a Segurança em **Java EE** antes de Java EE 8?
- Principais **novidades** da API de Segurança 1.0
  - Mecanismos de **autenticação**
  - **Identity Store**
  - **Contexto universal** de segurança

Quem sou eu? Who am I? Кто я?

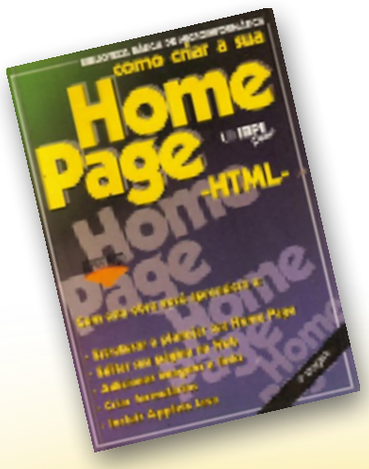
Helder da **Rocha**

Tecnologia * Ciência * Arte

HTML & tecnologias Web desde 1995

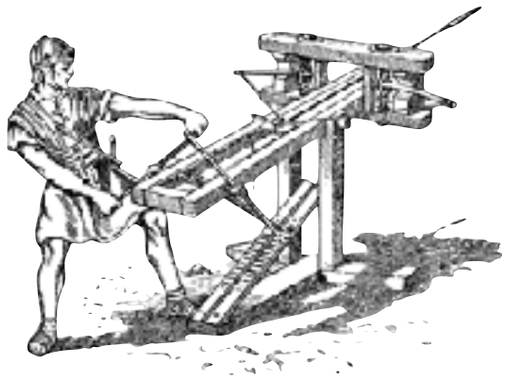
Autor de cursos e livros sobre

Java, XML e tecnologias Web



argonavis.com.br

helderdarocha.com.br



O que é **Segurança**?

- **Safety**: proteção contra danos **não-intencionais**
  - Problemas: **bugs** em geral e vulnerabilidades intrínsecas do sistema (threads, ambiente, recursos externos, etc.)
  - Proteção: qualidade da plataforma, boas práticas, testes, validação, monitoração, logging, checksums, etc.
- **Security**: proteção contra danos **intencionais**
  - Problemas: **ataques** em geral, acesso e manipulação de dados protegidos, danos ao sistema ou a terceiros
  - Proteção: autenticação, autorização, criptografia, auditoria, filtros, auditoria, hashes, etc.

Mecanismos de segurança
Java SE 8

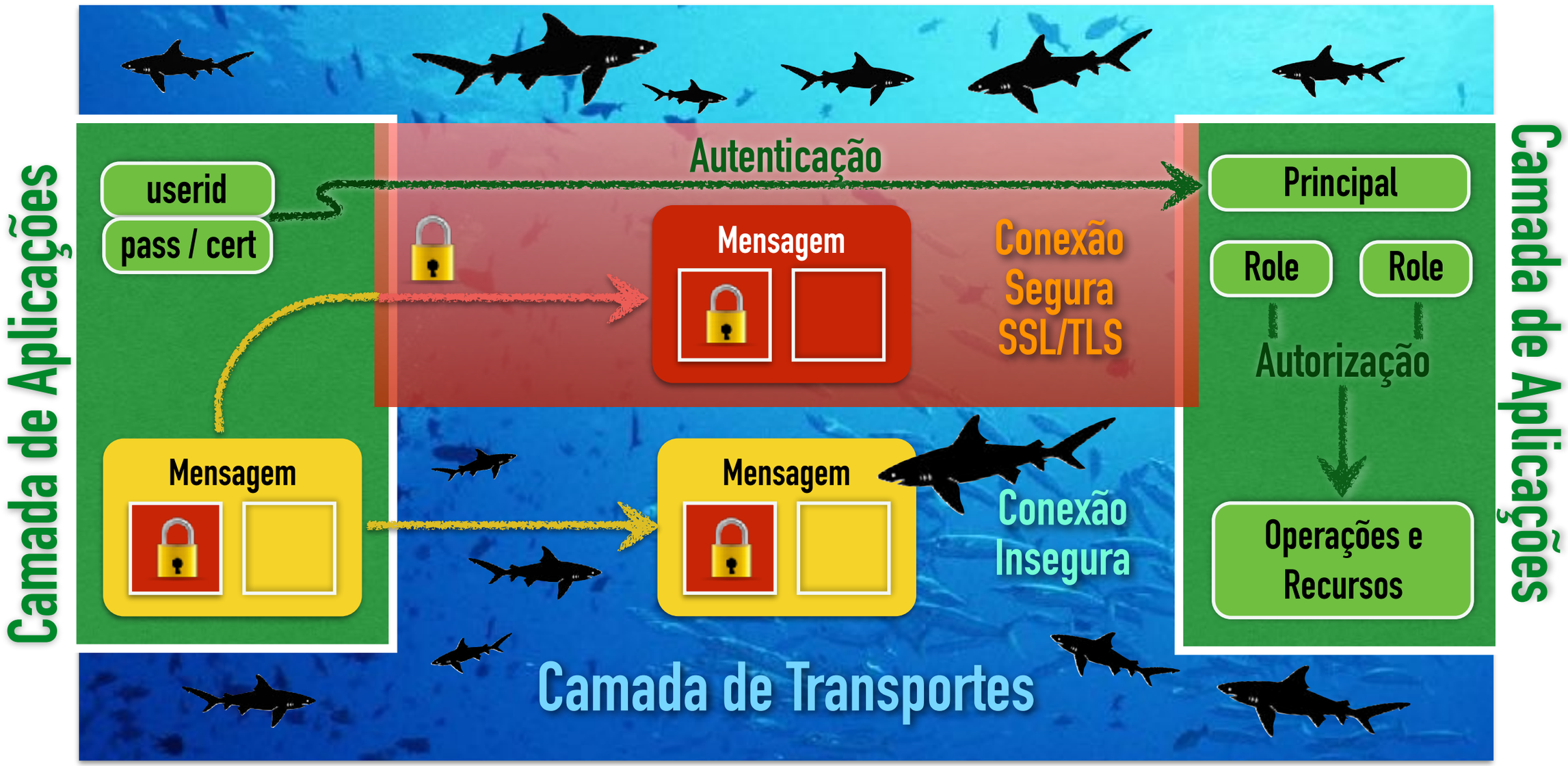
Segurança nativa da Plataforma Java

"Sandbox" da JVM
- Classloader
- Garbage collection
- Bytecode verification
- Data-typing

+

Criptografia	JCA JCE Java XML DSig (JSR 105)
Autenticação e Autorização	Policy / Security Manager JAAS JarSigner TSA/TSP (RFC 3161)
Infraestrutura de Chave Pública (PKI)	X.509, CRLs e CertPath API OCSP (RFC 2560) PKCS#11
Comunicação Segura	JSSE (SSL/TLS) SASL (RFC 2222) GSS-API (RFC 2853)

Mecanismos de segurança
Java EE 7

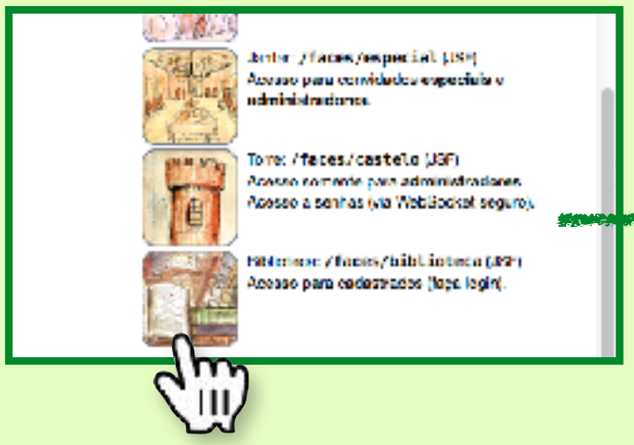


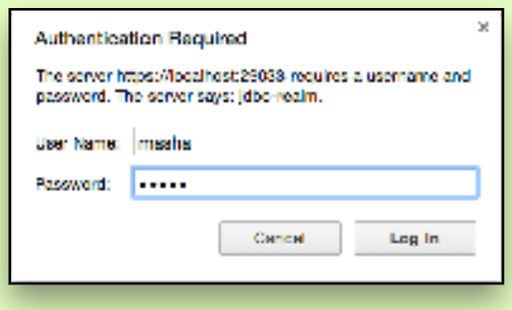
APIs programáticas: autenticação (**JASPIC**) e autorização (**JACC** + contextos de segurança de cada tipo de componente)

APIs declarativas: autorização, config de autenticação HTTP (JAAS) e config de transporte SSL/TLS por recurso/método HTTP

Autenticação & Autorização na Camada **Web**

Browser





HTTP

Requisição 1: Pede página restrita
GET /app/faces/biblioteca.xhtml

Resposta 1: Pede credenciais
401 Unauthorized
WWW-Authenticate: Basic realm="jdbc-realm"

HTTPS

HTTPS

Requisição 2: Envia credenciais
GET /app/faces/biblioteca.xhtml
Authorization: Basic bWFzaGE6MTIzNDU=

HTTPS

Resposta 2: Envia página restrita
200 OK

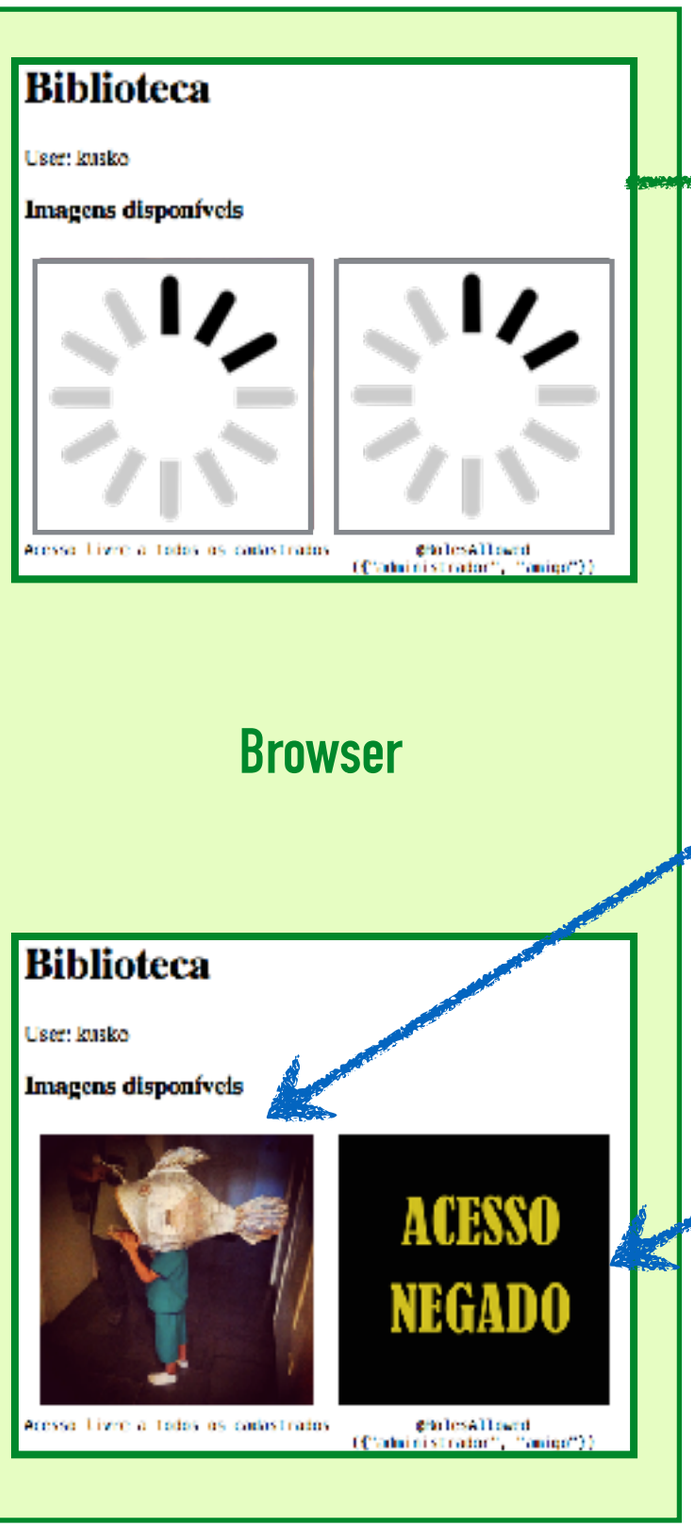
Web Container

Configuração (web.xml)
- HTTPS 2-way (CONFIDENTIAL)
- Autenticação: jdbc-realm
- Métodos: GET, POST
- Roles: todos os logados (**)

Autenticação
- Realizar autenticação
- Autenticado? Cria credencial.
  - Ir para Autorização
- Não autenticado?
  - Retorna 401 Unauthorized

Autorização
- Verificar permissões
- Autorizado?
  - Retorna recurso pedido
- Não autorizado?
  - Retorna 403 Forbidden

Autorização na Camada **EJB**



Req2

Res2

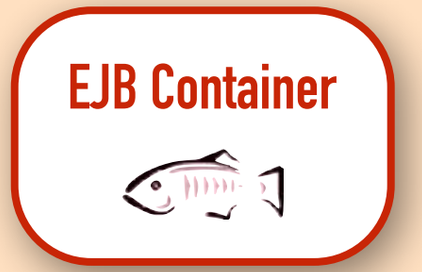
Contexto Web – Credenciais
Principal: kusko
Roles: {"outro"}

Servlet chama metodo 1 em EJB

```
try {
   b.getFigura1();
} catch (EJBAccessException e) {
   return "acessonegado.png";
}
```

Servlet chama metodo 2 em EJB

```
try {
   b.getFigura2();
} catch (EJBAccessException e) {
   return "acessonegado.png";
}
```



Contexto EJB (propagados)
Principal: kusko
Roles: {"outro"}

Retornar link

EJBAccessException

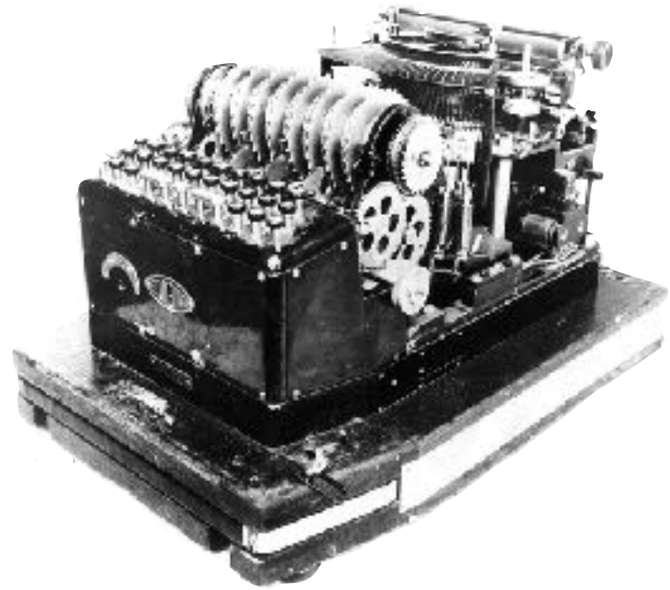
Autorização

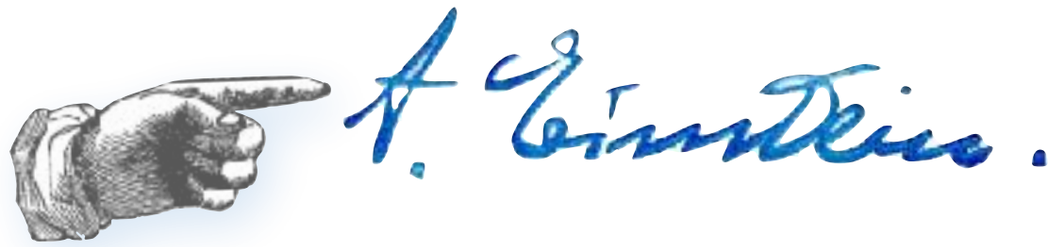
```
@RolesAllowed("**")
public String getFigura1() {
   return "peixinho.png";
}
```

```
@RolesAllowed("amigo")
public String getFigura2() {
   return "lupita.png";
}
```

Conexão **HTTP S**egura

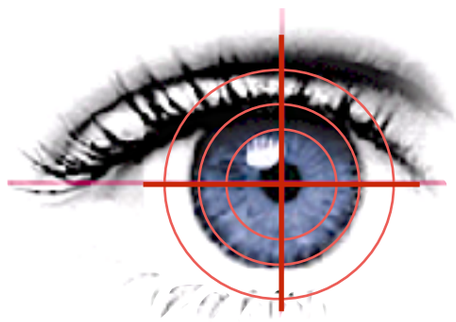
- Pode-se proteger recursos em Java EE **configurando** a necessidade do uso de HTTP + camada SSL/TLS
- SSL/TLS atua na **camada de transporte** protegendo contra principais riscos de transferir dados pela rede
  - Risco de não saber se outro **é quem diz que é**: **autenticação**
  - Risco da mensagem ser interceptada e **alterada**: **integridade**
  - Risco da mensagem ser interceptada e **lida**: **confidencialidade**
- SSL/TLS usa vários mecanismos de **criptografia**
  - Hashes
  - Criptografia de chave pública (assimétrica)
  - Criptografia de chave secreta (simétrica)
- Também protege **camada de mensagens** mas apenas durante a conexão





^63$t+p@55w0rd#

Autenticação



Autenticação

- Como **provar** que você é mesmo você?
  - Apresente **credenciais**: algo que você **tem** (ex: documento), **é** (ex: biometria) ou **sabe** (ex: senha)
- Autenticidade depende de
  - Parecer dado por uma **autoridade de confiança**
  - **Integridade** das informações: garantia que os dados transmitidos não foram alterados pelo caminho)

Principal

- **Representa** uma entidade autenticada
  - java.security.Principal
  - Contém sua **identificação** (nome / certificado) verificada através de credenciais
- Tem foco **diferente** em Java SE e Java EE
  - Em Java SE (**JAAS**) um **Subject** representa um usuário, que pode ter uma coleção de identidades (**Principal**)
  - Em APIs Java EE um **Principal** representa um usuário, que pode assumir uma coleção de papéis (**Roles**)

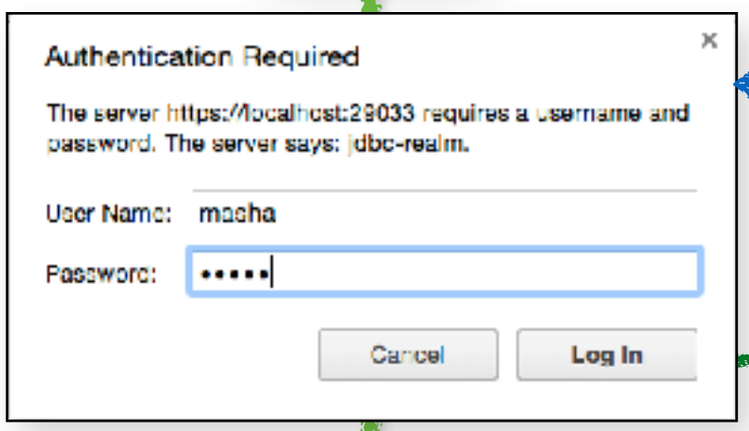
BASIC (RFC 2069 / 2617)

```
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>jdbc-realm</realm-name>
</login-config>
```

configuração
web.xml

GET /app/secret

401 Unauthorized
WWW-Authenticate: Basic realm="jdbc-realm"



Credenciais: encoding Base64

GET /app/secret
Authorization: Basic bWFzaGE6MTIzNDU=

200 OK

Para fazer logout: feche o browser!

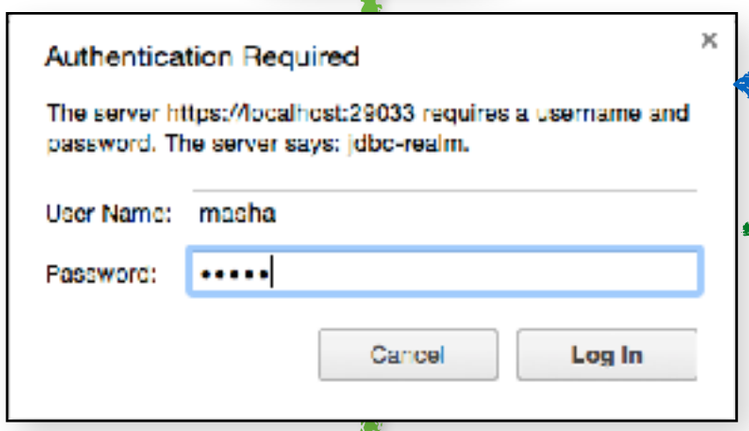
DIGEST (RFC 2069 / 2617)

```
<login-config>
   <auth-method>DIGEST</auth-method>
   <realm-name>jdbc-realm</realm-name>
</login-config>
```

configuração
web.xml

GET /app/secret

401 Unauthorized
WWW-Authenticate: Digest realm="jdbc-realm",
qop="auth", nonce="143...064", opaque="DF...5C"



GET /app/secret
Authorization: Digest username="masha", realm="jdbc-realm",
nonce="143...f89", uri="/app/faces/biblioteca.xhtml",
response="2c40...df", opaque="DF...5C", qop=auth,
nc=00000001, cnonce="66...948b"

Credenciais: MD5 hash

200 OK

Para fazer logout: feche o browser!

FORM

```
<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>
```

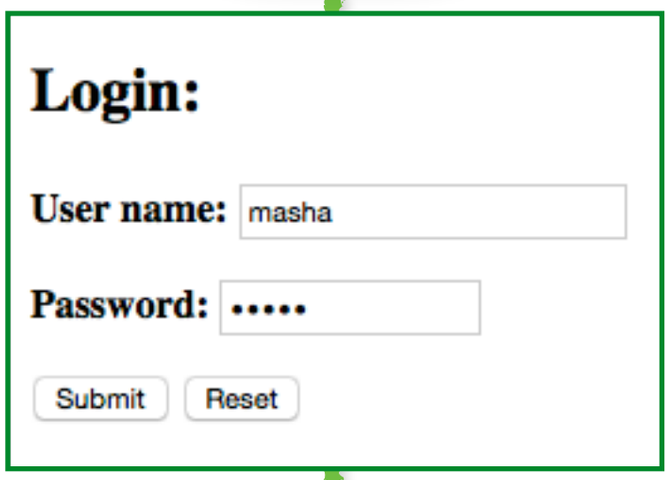
configuração
web.xml

GET /app/secret

200 OK

Set-Cookie: JSESSIONID=aac...44; Path=/app; Secure; HttpOnly

```
<form action="j_security_check"
      method="POST">
 <input type="text" name="j_username">
 <input type="password" name="j_password">
  <input type="submit">
</form>
```



POST https://localhost:29033/app/faces/j_security_check

Referer: https://localhost:29033/app/faces/biblioteca.xhtml

Cookie: JSESSIONID=aaab5...b1f6

Authorization:Basic Y2VyZWJybzoxMjM0NQ==

Encoding Base64

j_username:masha

j_password:12345

Proteção depende da camada SSL/TLS

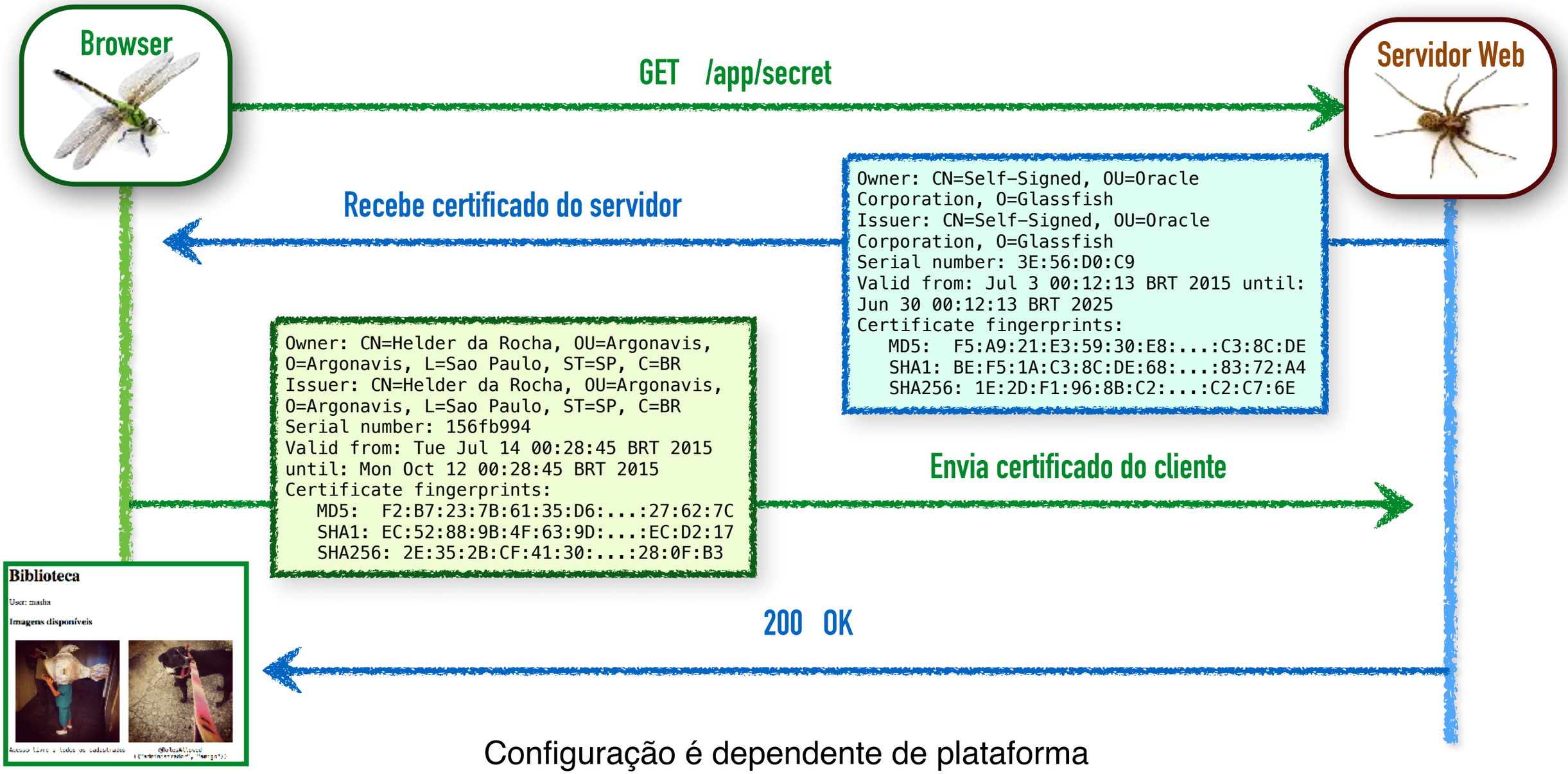
200 OK

Para fazer logout: **HttpSession#invalidate()**

CLIENT-CERT

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

configuração
web.xml



Configuração é dependente de plataforma

Autorização e role-mapping

Autorização

- Como **controlar o acesso** a recursos protegidos?
  - **Quem** pode acessar?
  - Quais **recursos**? URLs, componentes, classes, métodos, blocos de código, tabelas, registros, fragmentos de XML, imagens, arquivos, pastas, aplicações
  - Que **ações** podem executar? GET, POST, ler, gravar, criar, remover
  - Quais **condições**? "das 9 as 17h", "enquanto houver tokens", "apenas duas vezes"

Este tipo de controle de acesso não existe em Java EE 7

Grupos (~coleções de usuários **no servidor**)

- Usuários *podem* pertencer a **grupos**
  - Mas **usuários** não fazem parte do escopo do Java EE, e **grupos** também não
  - Conceito é dependente de plataforma

masha

masha

alienigenas

spock

marvin

megalomaniacos

cerebro

marvin

Grupos criados para aplicação exemplo no **Glassfish 4.1**

ratos

cerebro

niquel

russos

masha

vini

terraqueos

masha

vini

kusko

cerebro

niquel

Roles (~chaves de acesso a recursos protegidos)

- Principals (grupos e usuários) são **mapeados** a **roles**
  - Mapeamento padrão: responsabilidade do **identity store**

Exemplo de mapeamentos

role: **administrador** - grupo **alienigenas** - usuário **cerebro**	
role: **especial** - usuário **vini** - usuário **masha**	
role: **amigo** - usuário **vini** - usuário **masha** - usuário **niquel**	
role: **outro** - usuário **kusko**	

Autorização em Java EE

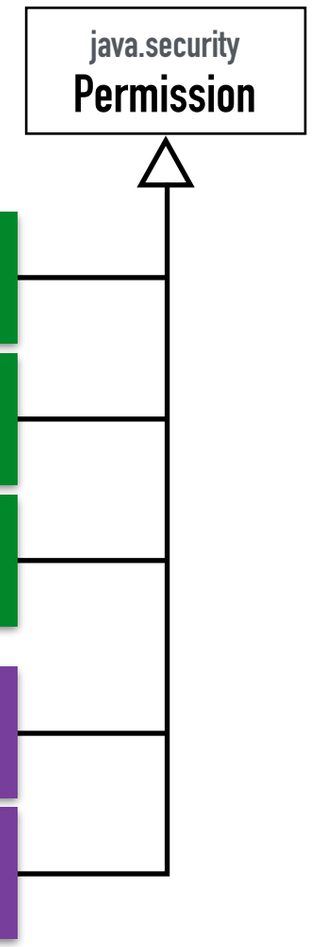
- Baseada em **roles**, fácil de usar e 100% Java EE
  - Configuração **declarativa** via anotações e deployment descriptor XML ou anotações
  - Acesso (leitura) através APIs programáticas em **contextos de segurança**

	Anotações	Deployment descriptors

Camada Web	`@ServletSecurity(`
	`  @HttpConstraint(`
	`  transportGuarantee = ...,`
	`  rolesAllowed = {...}`
	`  )`
	`)`
	`public class MyServlet... {...}`
		web.xml
		`<security-constraint>`
		`  <web-resource-collection>`
		`  ...`
		`  </web-resource-collection>`
		`  <auth-constraint>`
		`  <role-name>...</role-name>`
		`  </auth-constraint>`
		`  <user-data-constraint>`
		`  <transport-guarantee>...</transport-guarantee>`
		`  </user-data-constraint>`
		`</security-constraint>`

Camada EJB	`@RolesAllowed({...})`
	`public class Bean {...`
	`  @DenyAll`
	`  public void m1() {}`
	`  @PermitAll`
	`  public void m2() {}`
		ejb-jar.xml
		`<method-permission>`
		`  <role-name>...</role-name>`
		`  <method>`
		`  <ejb-name>...</ejb-name>`
		`  <method-name>...</method-name>`
		`  </method>`
		`</method-permission>`

JACC (JSR 115)



Autorização em Webapps

```
@WebServlet(name = "ServletSecreto", urlPatterns = {"/secreto"})
@ServletSecurity(
   @HttpConstraint(
         transportGuarantee = TransportGuarantee.CONFIDENTIAL,
         rolesAllowed = {"amigo", "administrador"}
   )
)
public class ServletSecreto extends HttpServlet { ... }
```

```
@WebServlet(name = "LabirintoServlet", urlPatterns = {"/labirinto"})
@ServletSecurity(httpMethodConstraints={
   @HttpMethodConstraint("GET"),
   @HttpMethodConstraint(value="POST", rolesAllowed={"outro"}),
   @HttpMethodConstraint(
         value="TRACE",
         transportGuarantee = TransportGuarantee.NONE,
         rolesAllowed = {"amigo", "administrador"}
   )
})
public class LabirintoServlet extends HttpServlet { ... }
```

Autorização em EJB

```
@DeclareRoles({"amigo", "administrador"})
@Stateless public class ImagemBean {
    @RolesAllowed("administrador")
    public void removerUsuario(Usuario u) {  ... }

    @RolesAllowed("amigo")
    public List<Usuario> listaDeUsuarios() { ... }

    @PermitAll
    public String getInfo() { ... }

    @RolesAllowed("* *")
    public String getFreeStuff() { ... }

    @DenyAll
    public String loggingData() { ... }
}
```

Como implementar?

- Até Java EE 7
- **Autenticação**
  - Requer o uso de ferramentas **proprietárias** ou escrever um LoginModule usando JAAS (Java SE) com configuração **proprietária**
  - Autenticação programática (JASPIC) é **complexa** e não suportada em todos os servidores
- **Autorização**
  - Autorização declarativa via deployment descriptors ou anotações, com declaração de roles
  - Configuração de roles **proprietária** (identity store fornecido pelo servidor e mapeado via configuração **proprietária** à aplicação)

Autenticação via **HttpServletRequest**

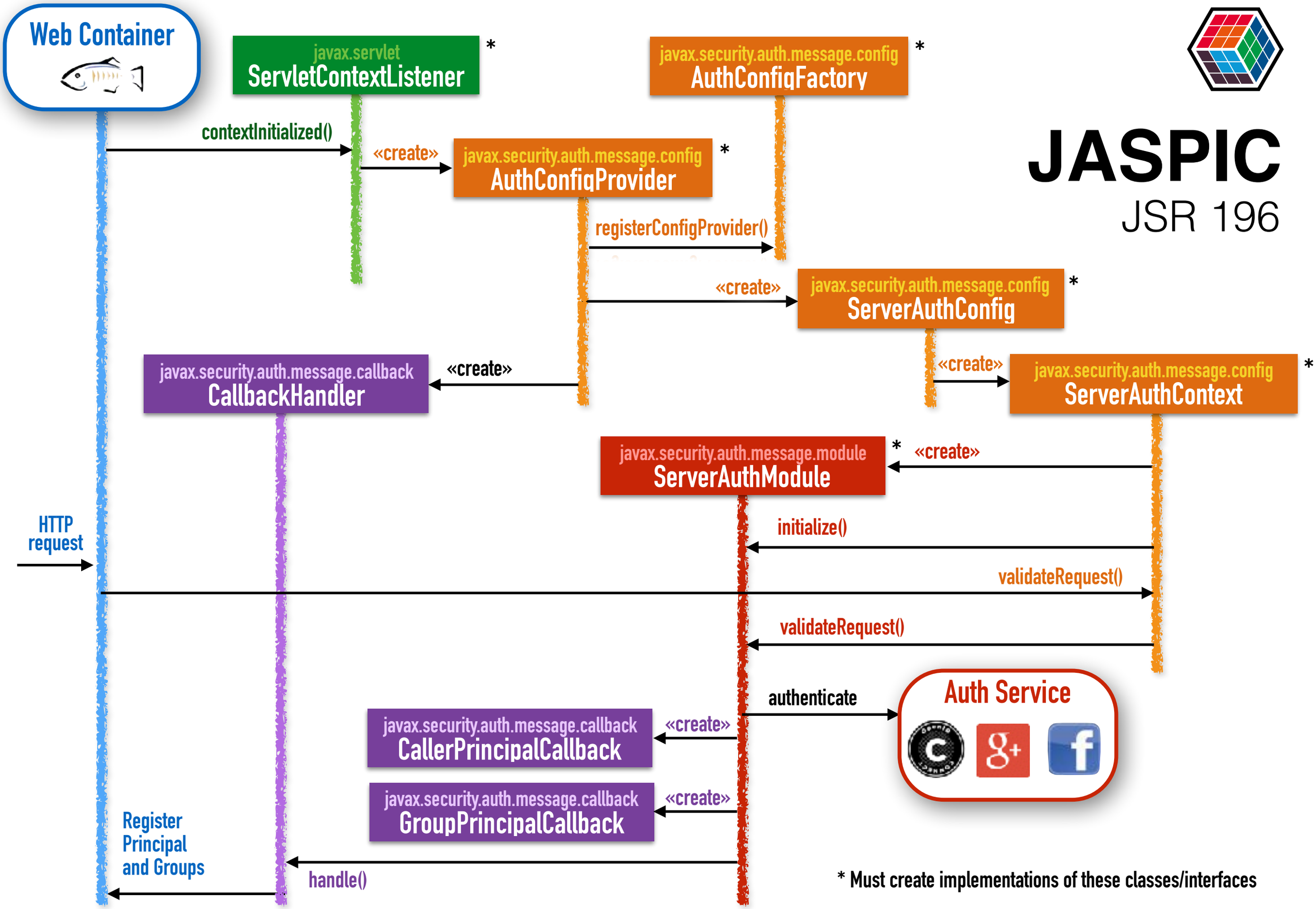
- API programática (facilita a criação de filtros)
- **`getAuthType`**(): String
  - Retorna string com mecanismo de autenticação (`BASIC`, `DIGEST`, `FORM`, `CLIENT-CERT`) ou `null`
- **`authenticate`**(HttpServletResponse response)
  - Autenticar usando mecanismo configurado
- **`login`**(String nome, String senha)
  - Autenticar usando login e senha
- **`logout`**()
  - Faz com que `getRemoteUser()`, `getCallerPrincipal()` e `getAuthType()` retorne `null`

Autenticação usando JASPIC **SAM (ServerAuthModule)**

```
public interface ServerAuthModule {

    public void initialize(...) throws AuthException;

    public AuthStatus validateRequest(...);

    public Class<?>[] getSupportedMessageTypes();

    public AuthStatus secureResponse(...);

    public void cleanSubject(...);

}
```

É preciso implementar várias classes e interfaces!

JASPIC
JSR 196



* Must create implementations of these classes/interfaces

Objetivos da JSR-375

Simplificar, padronizar e modernizar

"This JSR will holistically attempt to **simplify**, **standardize**, and **modernize** the Security API across the platform in areas identified by the community via survey results and submitted JIRA issues."

Propostas do JSR 375

- Terminologia unificada
- **API de autenticação**
- **API de identity store**
- API de password aliasing
- API de atribuição de roles e permissões
- API de interceptadores de autorização
- **API de contexto de segurança**

Objetivos alcançados na versão 1.0

Soteria

- Implementação de referência do Java EE 8 Security 1.0
- Pode ser usado em sistemas Java EE 7 importando os JARs:

```
<dependency>
  <groupId>org.glassfish.soteria</groupId>
  <artifactId>javax.security.enterprise</artifactId>
  <version>1.0</version>
</dependency>
```

Java EE Security 1.0

- **javax.security.enterprise**
  - **SecurityContext** (interface), **CallerPrincipal** (classe), **AuthenticationStatus** (enum), AuthenticationException
- **j.s.e.authentication.mechanism.http**
  - Classes, interfaces e anotações para **autenticação**
- **j.s.e.credential**
  - Interface Credential, classe Password e implementações de **Credential**
- **j.s.e.identitystore**
  - API para **identity store**

javax.security.enterprise

SecurityContext

AuthenticationStatus

CallerPrincipal

AuthenticationException

javax.security.enterprise

SecurityContext

AuthenticationStatus

SUCCESS
NOT_DONE
SEND_FAILURE
SEND_CONTINUE

CallerPrincipal

AuthenticationException

Anotações de autenticação

@BasicAuthenticationMechanism (BASIC)

- realmName="" (WWW-Authenticate - não é ID store)

@FormAuthenticationMechanism (FORM) **e**
@CustomFormAuthenticationMechanism (authenticate())

- loginToContinue=@LoginToContinue()

@LoginToContinue

- errorPage="/login-error", loginPage="/login", useForwardToLogin=true, useForwardToLoginExpression=""

@RememberMe, @AutoApplySession

Exemplos

- 1. BASIC:

```
@BasicAuthenticationMechanismDefinition(realmName="user-realm")
@WebServlet("/login")
@DeclareRoles({ "admin", "user"})
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
public class LoginServlet extends HttpServlet { ... }
```

- 2. FORM:

```
@FormAuthenticationMechanismDefinition(
   loginToContinue = @LoginToContinue(
        loginPage = "/login",
        errorPage = "/login-error"))
@ApplicationScoped
public class ApplicationConfig { ... }
```

3. Form com authenticate()

```
@Named @RequestScoped
public class LoginBean {

  @Inject SecurityContext sctx;
  String user, pass;

  public void login() {
    Credential credential =
        new UsernamePasswordCredential(user, new Password(pass));
    AuthenticationStatus status = securityContext.authenticate(
      request, response, withParams().credential(credential));

    if (status.equals(SEND_CONTINUE))
      // SUCESSO
     else if (status.equals(SEND_FAILURE))
      // FALHA
  } ...
}
```

```
@CustomFormAuthenticationMechanismDefinition(
  loginToContinue = @LoginToContinue(
    loginPage="/login.xhtml"
  )
)
@WebServlet("/admin")
@DeclareRoles({ "admin", "user"})
@ServletSecurity(@HttpConstraint(rolesAllowed = "admin"))
public class AdminServlet extends HttpServlet { ... }
```

```
<form jsf:id="form">
    <b>Userid</b>
    <input jsf:id="user" type="text"
           jsf:value="#{loginBean.user}" />
    <b>Senha</b>
    <input jsf:id="pass" type="password"
           jsf:value="#{loginBean.pass}" />
    <input type="submit" value="Login"
           jsf:action="#{loginBean.login}" />
</form>
```

Java EE 8 - Interface **HttpAuthenticationMechanism**

- Para uso em servlet endpoints (ex: JAX-RS, JSF)
  - É uma versão do **ServerAuthModule** (JASPIC) compatível com CDI e específica para servlets
  - Métodos recebem **HttpServletRequest**, **HttpServletResponse** e **HttpMessageContext**
  - **Liberdade total** para registrar autenticação! (pode ou não delegar para um Identity Store)
  - HttpMessageContext:: **notifyContainerAboutLogin**(Principal p, Set<String> roles)

4. Autenticação JASPIC

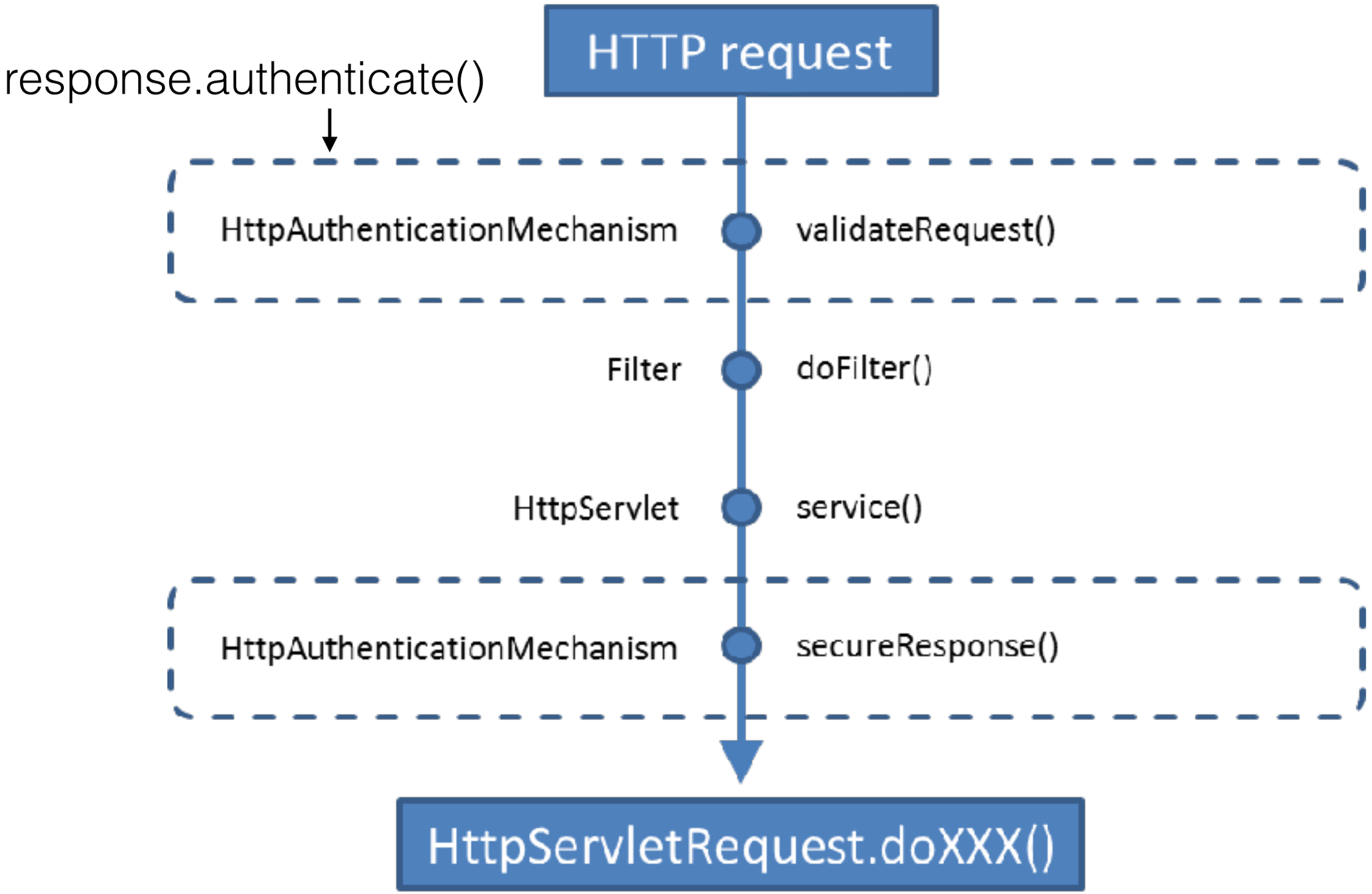
- Implemente **HttpAuthenticationMechanism** (HAM) como um bean CDI com **ApplicationScope**
- Implemente os métodos

`AuthenticationStatus` **`validateRequest`**
`(HttpServletRequest, HttpServletResponse, HttpMessageContext)`
pode executar **qualquer** autenticação e retorna o status

`secureResponse()`
default não criptografa.

`cleanSubject()`
para **logoff** - *default*

Processo de execução



Fonte: IBM developerworks

@RememberMe

- Usada na implementação de um HAM para que a autenticação de um usuário seja lembrada e aplicada a cada requisição

```
@RememberMe(
    cookieMaxAgeSeconds = 1800
)
@ApplicationScoped
public class MyAuthenticationMechanism
    implements HttpAuthenticationMechanism { ... }
```

- Atributos
  - cookieHttpOnly=true, cookieMaxAgeSeconds=86400, cookieSecureOnly=true, isRememberMe=true, *Expression="", cookieName=JREMEMBERMEID
- **@AutoApplySession**: aplica automaticamente a sessão para o usuário logado

Identity Store

A. Einstein.

Usuários (pessoas, máquinas, sistemas)


spock


masha


vini


kusko


marvin


cerebro


niquel

- A criação e autenticação de usuários (principais) é responsabilidade da **identity store**

Gerência de usuários: Java EE 7

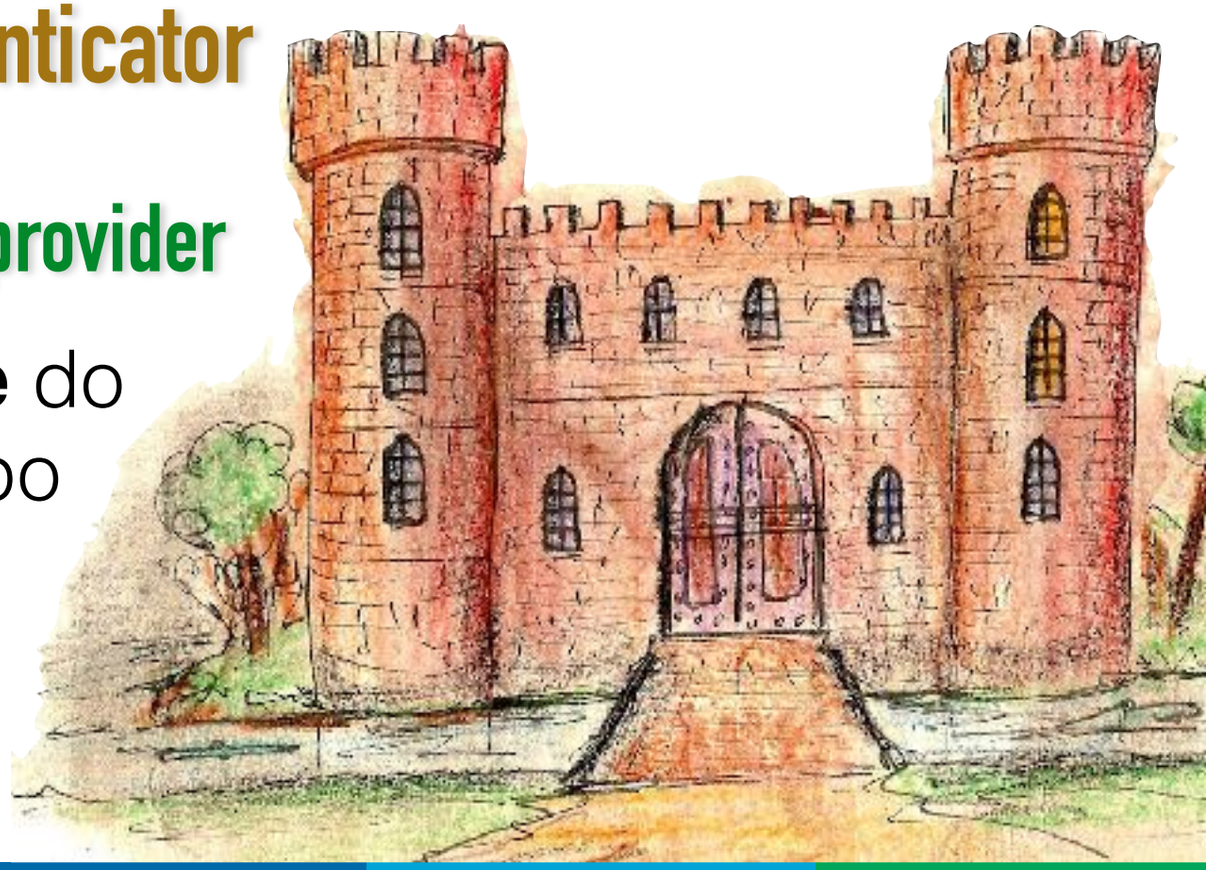
- **Não existe suporte padrão** para gerenciar usuários em Java EE 7
- Aplicações não tem como **criar**, **remover**, **atualizar** e **agrupar** usuários em Java EE
- Dependência de **soluções proprietárias** (vendor-lock-in), **frameworks** de terceiros ou soluções **in-house**

Realm (~reino, domínio)

- **Autoridade** que administra **usuários e grupos** e que determina **escopo** de políticas de segurança (ex: LDAP, banco de dados, banco de certificados, FB, Google)
- Também chamado de:

zone
security provider
identity store
identity manager
login module
domain
authenticator
identity provider
auth store
region
auth repository
auth provider

- Java EE utiliza apenas o **nome** do realm (para selecionar o escopo da autenticação via HTTP)

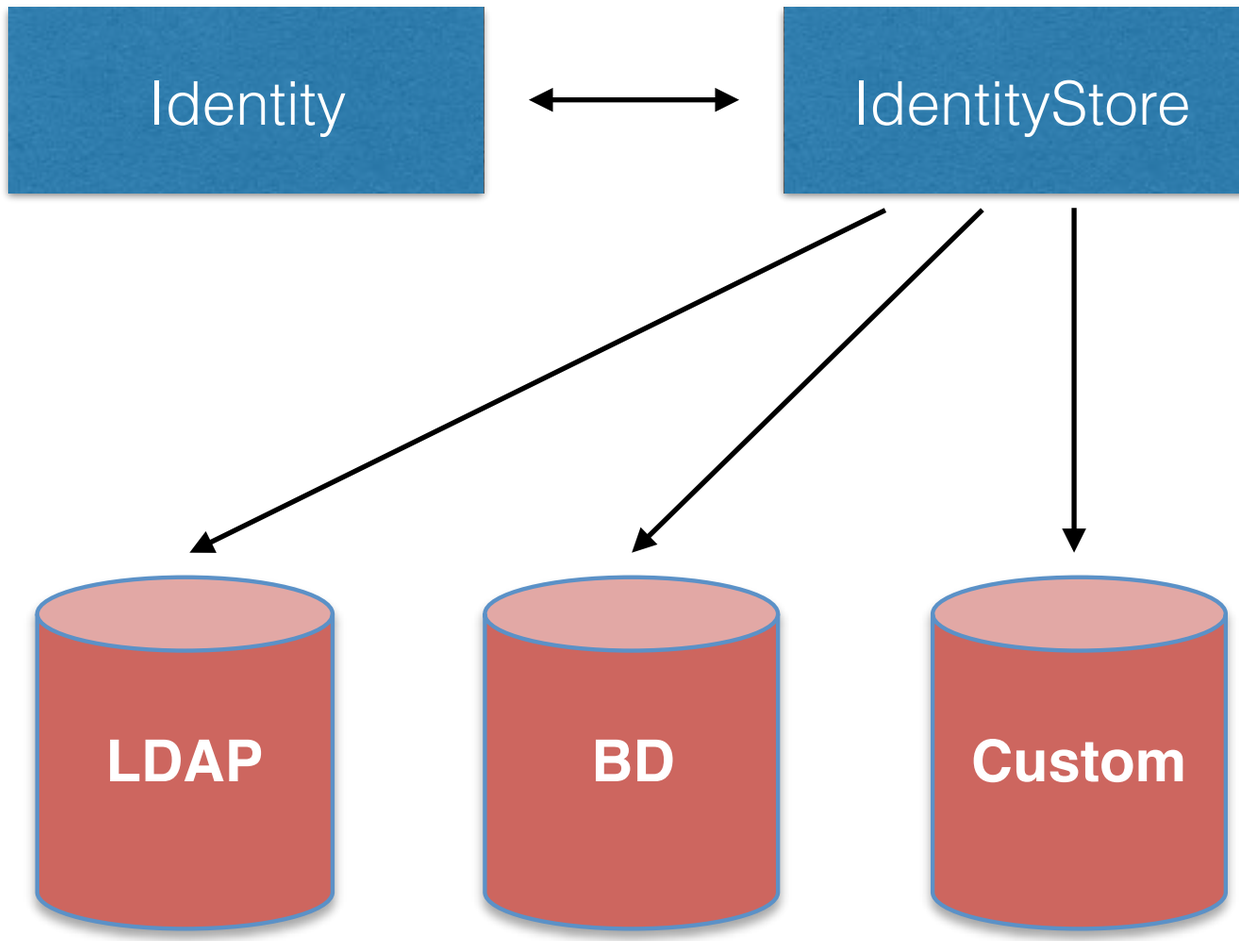


Proposta JSR-375: **Identity Store**

- **Serviço padrão** para criar, atualizar, remover, e agrupar usuários
  - Usa um **recurso de usuários e grupos** (banco de dados ou LDAP)
    - Poder ser intercambiável: recurso diferente para desenvolvimento, teste, produção
- Interface IdentityStore implementada em Java EE 8 / Soteria
- Injetado em CDI através do **IdentityStoreHandler**

```
@Inject
private IdentityStoreHandler idStoreHandler;
```

Identity Store: implementação



Anotações de IdentityStore

- **@DatabaseIdentityStoreDefinition**
  - callerQuery="",
  - groupsQuery="",
  - dataSourceLookup="java:comp/DefaultDataSource",
  - hashAlgorithm= javax.security.enterprise.identitystore.Pbkdf2PasswordHash.class,
  - hashAlgorithmParameters={}, priority=70,
  - useFor={IdentityStore.ValidationType.VALIDATE, IdentityStore.ValidationType.PROVIDE_GROUPS},
  - *Expression
- **@LdapIdentityStoreDefinition**

Identity Store: exemplo usando banco de dados

```
@DatabaseIdentityStoreDefinition(
    dataSourceLookup = "java:global/permissions_db",
    callerQuery = "select senha from usuarios where nome = ?",
    groupsQuery = "select grupo from grupos where usuario = ?",
    hashAlgorithm = PasswordHash.class,
    priority = 10
)
@ApplicationScoped
@Named
public class ApplicationConfig { ... }
```

grupos

usuario	grupo

usuarios

nome	senha

Custom IdentityStore

- IdentityStores customizados (arquivo, cookies, memória, etc.) - ideal para testes
- Implementar os métodos (todos são default)
  - CredentialValidationResult **validate(Credential)** - deve retornar CredentialValidationResult.**VALID** ou **INVALID**
  - Set<String> **getCallerGroups(CredentialValidationResult)** - retorna os grupos associados ao principal
  - int **getPriority()**
  - Set<IdentityStore.ValidationType> **validationTypes()**

Contexto de segurança

Como obter **principals** e testar **roles**

em Java EE 7

WebServlets, Facelets, WebFilters

getUserPrincipal()
isUserInRole()

javax.servlet.http.
HttpServletRequest

EJBs

getCallerPrincipal()
isCallerInRole()

javax.ejb.
EJBContext

SOAP Web Services

getUserPrincipal()
isUserInRole()

javax.xml.ws.
WebServiceContext

CDI

@Inject
java.security.Principal

JSF backing beans

getUserPrincipal()
isUserInRole()

javax.faces.context.
ExternalContext

WebSockets

getUserPrincipal()

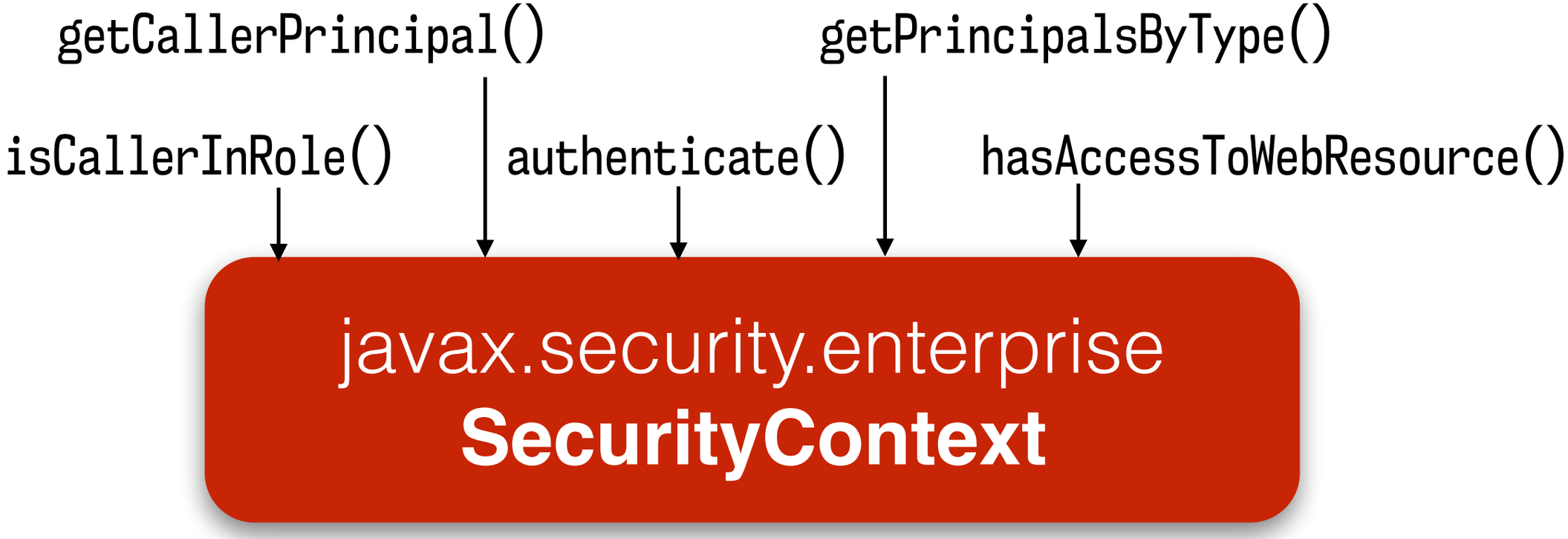
javax.websocket.
Session

RESTful Web Services

getUserPrincipal()
isUserInRole()

javax.ws.rs.core.
SecurityContext

SecurityContext em Java EE 8



Interface **SecurityContext**

- Pode ser **injetada** (CDI) em componentes Java EE para acesso programático à API de segurança
- Métodos
  - AuthenticationStatus **authenticate**(req, res, params)
  - java.security.Principal **getCallerPrincipal**()
  - Set<Principal> **getPrincipalsByType**(Class type)
  - boolean **isCallerInRole**(role)
  - boolean **hasAccessToWebResource**(res, mets...)

Java EE Security 1.0: contexto de segurança

- Um contexto de segurança pode ser obtido por **qualquer** componente Java EE

```
public class MyCdiBean {
    @Inject
    private SecurityContext securityContext;

    public String sayHello() {
        if (securityContext.isCallerInRole("admin")) {
            return "Hello World!";
        }
        throw new SecurityException("User is unauthorized.");
    }
}
```

Conclusões

- Java EE Security 1.0 ainda não solucionou todos os problemas de segurança do Java EE
- Mas conseguiu **simplificar** e **padronizar** recursos-chave da API
  - **Autenticação** independente de fabricante e configurável via anotações
  - **Identity Store** independente de fabricante e facilmente configurável, injetável via CDI
  - **Contexto de Segurança** universal, injetável e único para todos os componentes

Baixe esta palestra em

http://www.argonavis.com.br/download/tdc_2018_flo_javaee_security.html

Código disponibilizado no GitHub:

https://github.com/argonavisbr/JavaEE7SecurityExamples
https://github.com/argonavisbr/JavaEE8SecurityExamples

helder da rocha
helder@summa.com.br