JSR 375

O futuro da segurança em **Java EE 8**

helderdarocha

helder@argonavis.com.br

Objetivos

Discutir os **problemas** que existem na API padrão de segurança do Java EE atual

Apresentar as **idéias** que estão sendo discutidas sobre o **JSR-375:** API de Segurança do Java EE 8

Quem sou eu? Who am I? Кто я?

Helder da **Rocha**

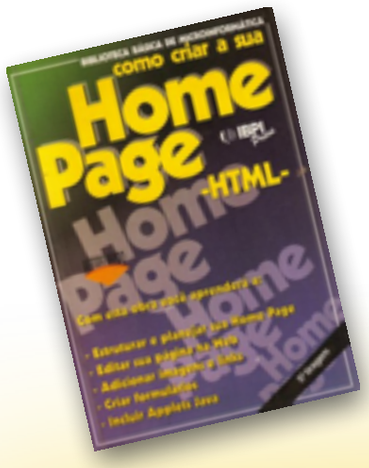
Tecnologia * Ciência * Arte

Java & Web desde 1995

Autor de cursos e livros

\+ de 2500 alunos

\+ de 8000 horas de aula



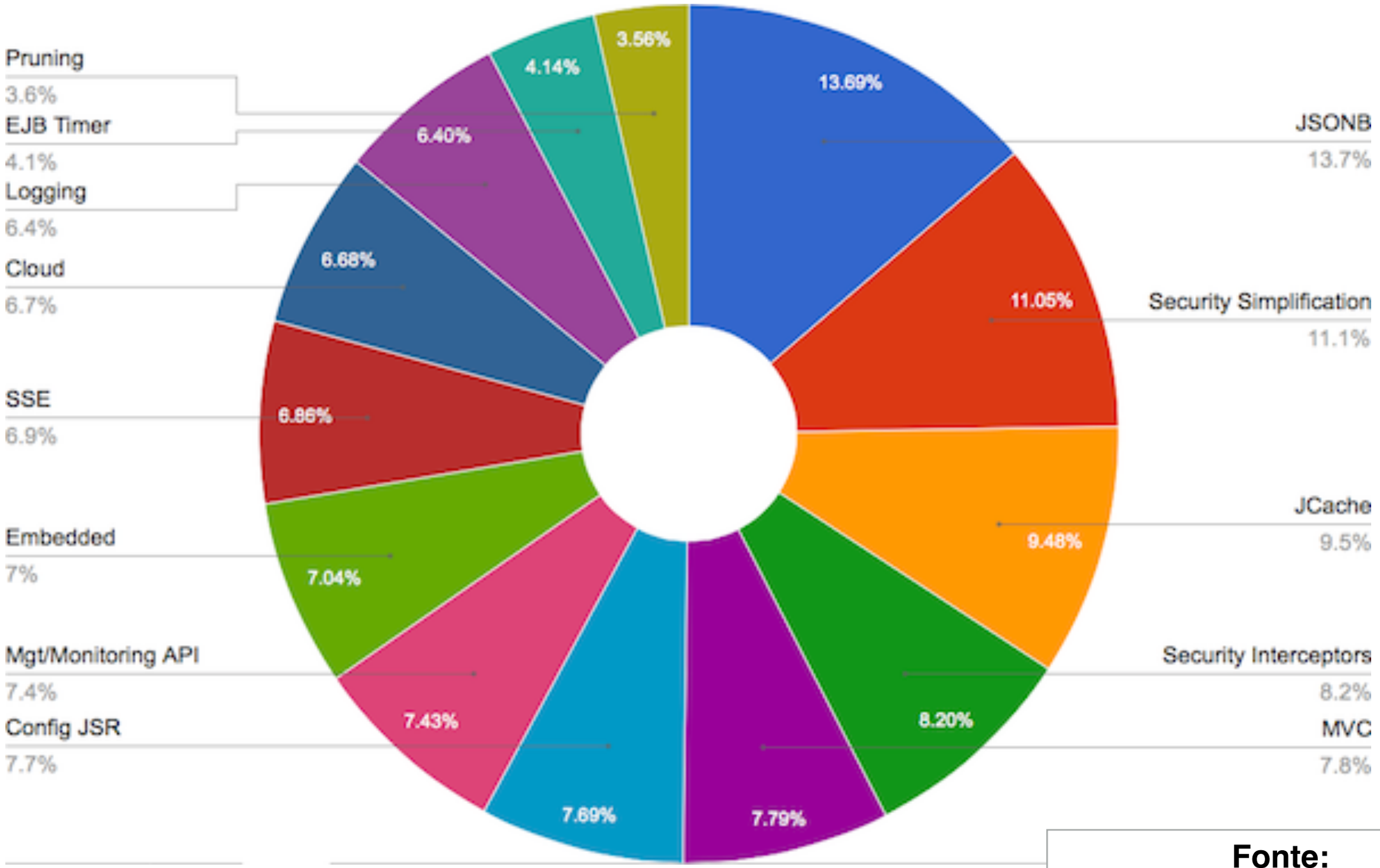
argonavis.com.br

helderdarocha.com.br

A API padrão de segurança do Java EE é muito **complexa**?

Java EE 8 Community Survey

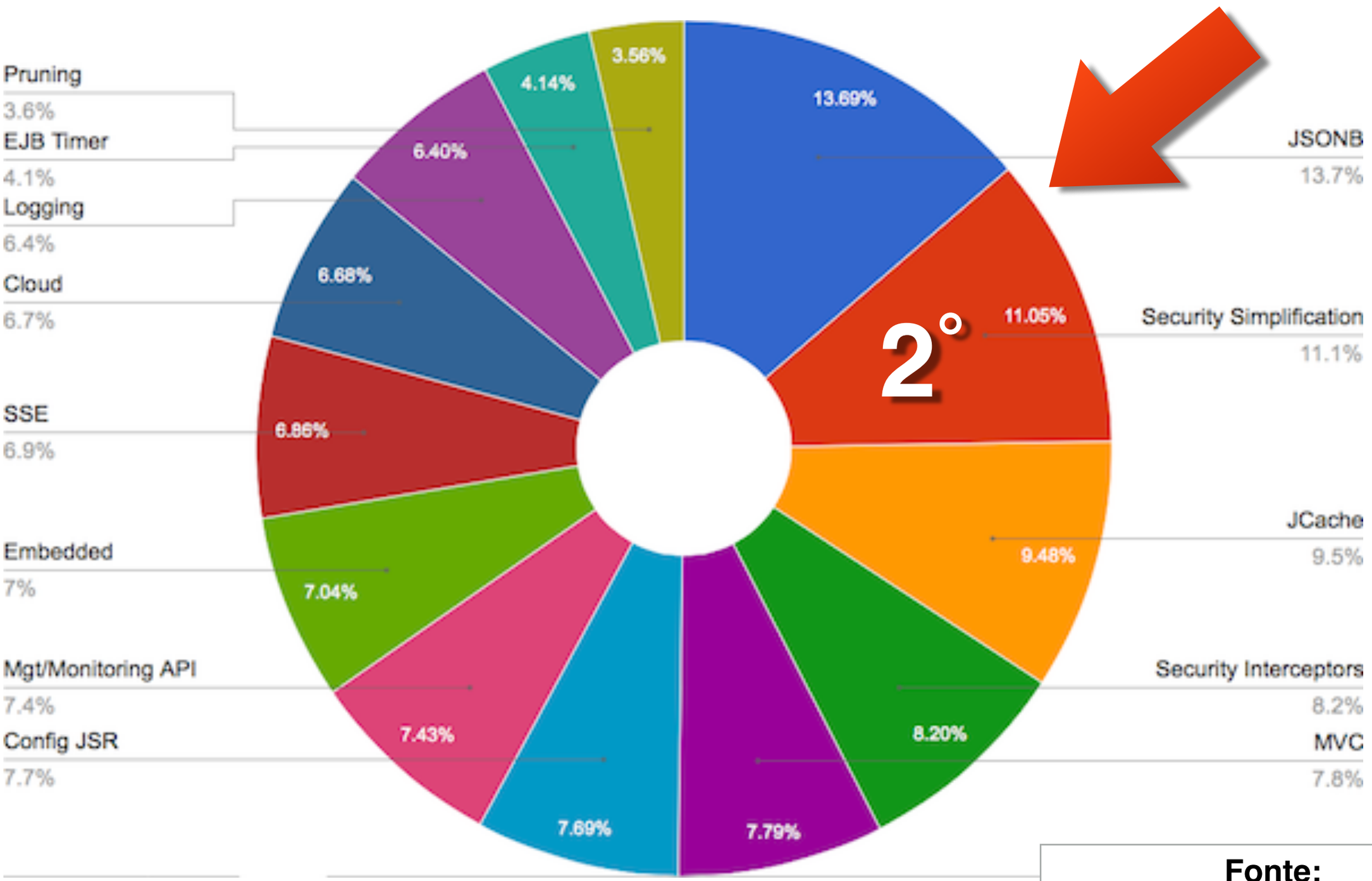
O que deve ser priorizado?



Fonte:
Java EE 8 Community Survey

Java EE 8 Community Survey

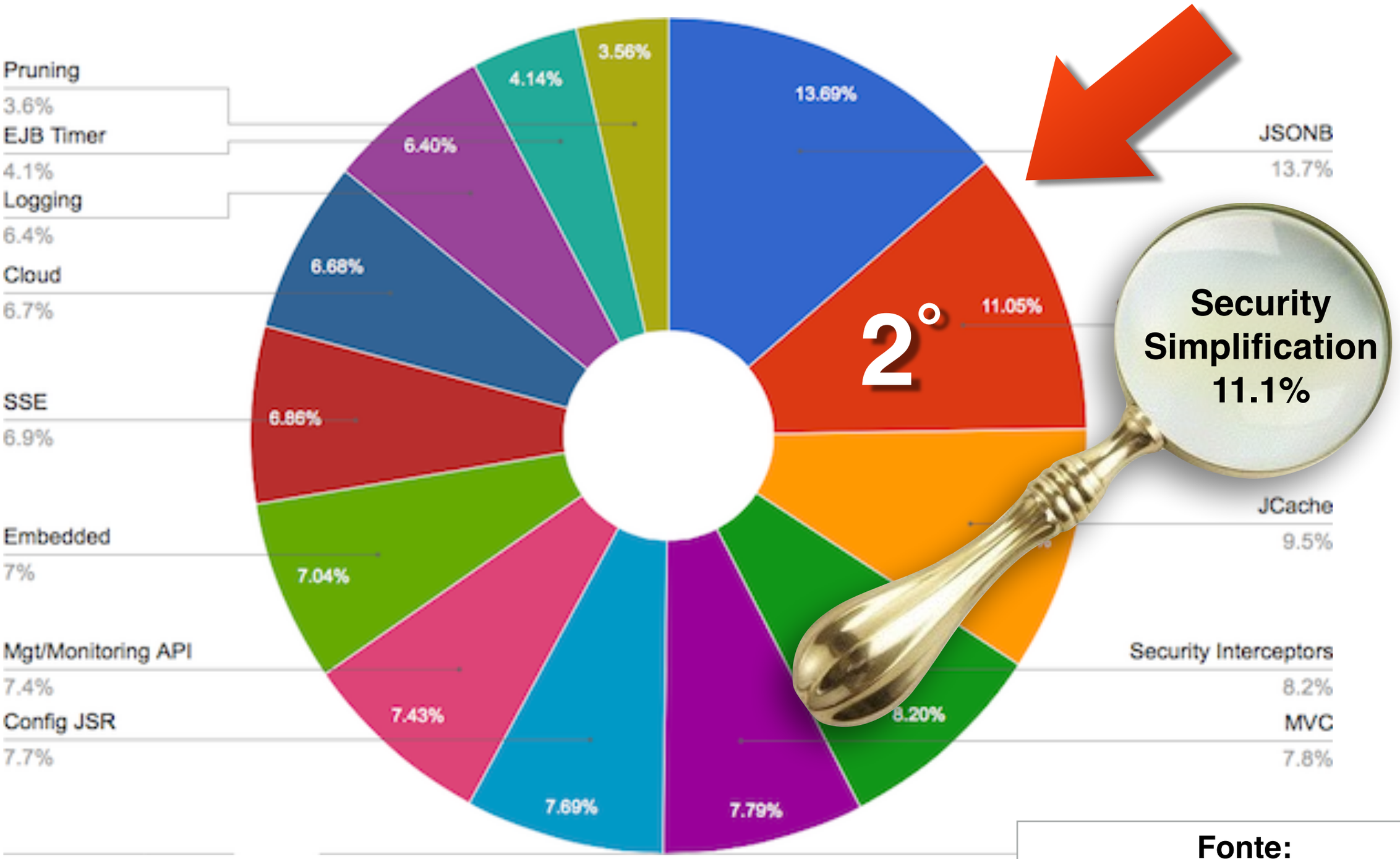
O que deve ser priorizado?



Fonte:
Java EE 8 Community Survey

Java EE 8 Community Survey

O que deve ser priorizado?



Fonte:
Java EE 8 Community Survey

Quais são os problemas?

- É **complexa**, confusa, difícil de usar
  - Redundância desnecessária
  - Volta aos tempos do J2EE
- Precisa ser **modernizada**
  - Não suporta OpenID Connect, OAuth, etc
  - Difícil de usar na nuvem e PaaS

Como mapear usuários no **Tutorial Java EE 7**

É possível implementar uma solução completa de segurança em Java EE que não dependa de APIs ou configurações proprietárias ou frameworks de terceiros?

Quais são os problemas?

- É **complexa**, confusa, difícil de usar
  - Redundância desnecessária
  - Volta aos tempos do J2EE
- Precisa ser **modernizada**
  - Não suporta OpenID Connect, OAuth, etc
  - Difícil de usar na nuvem e PaaS
- **Não existe** API padrão Java EE para segurança!
  - **Vendor lock-in** com solução do fabricante
  - Desenvolvedores abandonam Java EE em busca de soluções de terceiros (Spring, Shiro, etc.)

Problemas da segurança Java EE

- Necessidade de usar configurações proprietárias
- Em ambiente de nuvem/PaaS é problema: desenvolvedores não tem acesso a recursos proprietários - não se adapta ao paradigma do desenvolvedor de apps para a nuvem
- API não é portável, é confusa, antiquada
- Desenvolvedores migram para frameworks de terceiros já que o padrão não existe

Segurança **padrão** no Java EE

- Vendor lock-in: escolha a sua prisão

Mecanismos de segurança
Java SE 8

Segurança nativa da Plataforma Java

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

+

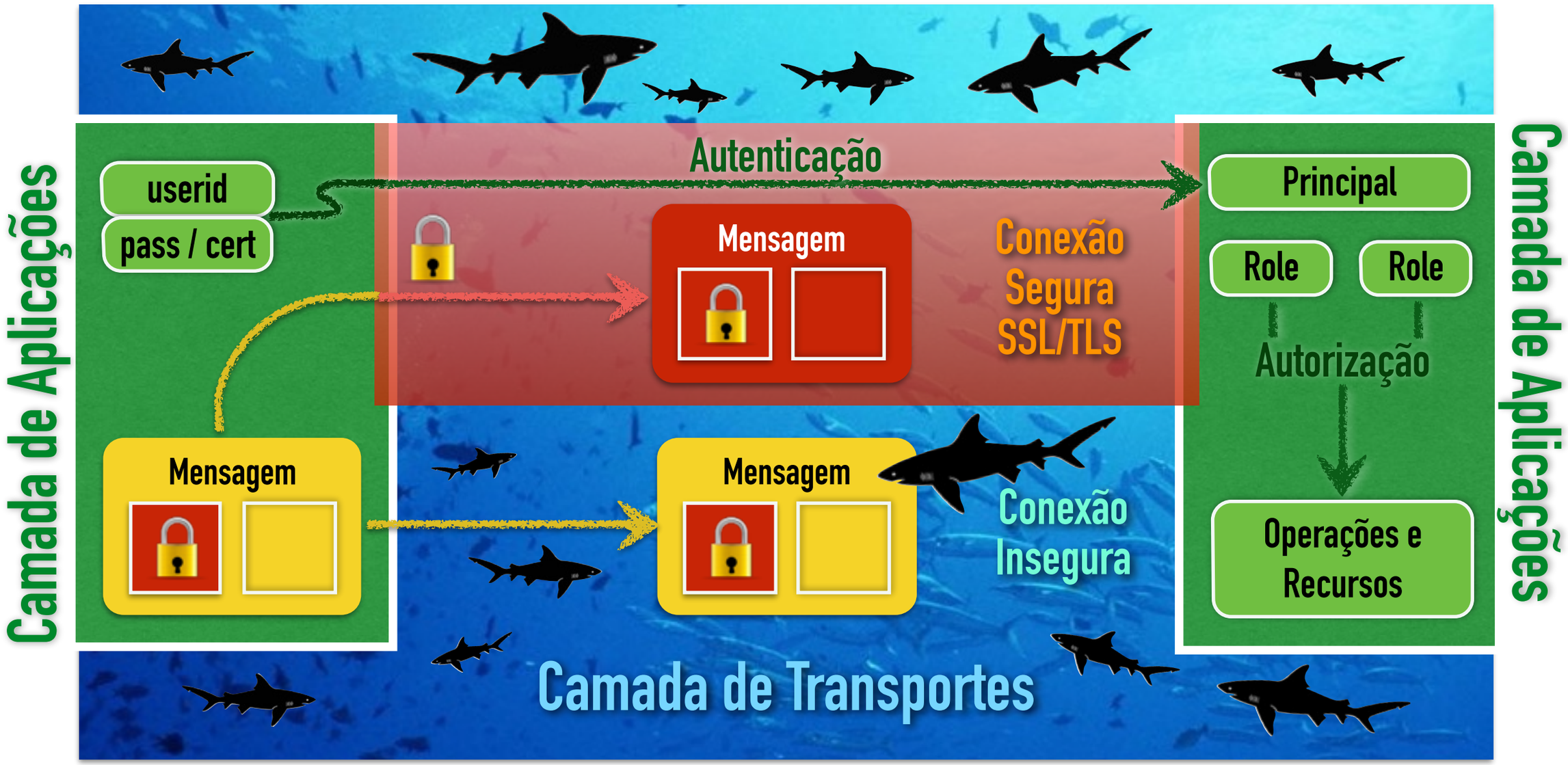
Criptografia
- JCA
- JCE
- Java XML DSig (JSR 105)

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

Segurança

O que falta em Java EE?

- **Ter uma API padrão de segurança** moderna, unificada e independente de fabricante
  - Segurança Java EE hoje, na prática, usa recursos proprietários ou frameworks de terceiros (Spring, etc.)
- O que mais?
  - **Terminologia** comum (callers/users, groups/roles, identity store)
  - Contexto de segurança acessível de **qualquer componente**
  - Mecanismo de autenticação mais **flexível** , **independente de servidor** e com implementações **padrão**
  - APIs **padronizadas** para identity stores (não depender de realms proprietários)
  - Formas de **controlar acesso** além de roles (regras, interceptadores)
- **Como será o futuro da segurança em Java EE?**

JSR 375

O que fazer

- Resolver problemas de **portabilidade**
- **Facilitar** a vida do desenvolvedor
  - Usar **defaults** e não precisar XML
  - Configuração **independente de servidor**
- **Modernizar**
  - Atender demandas **atuais** de segurança
  - **Inspiração** em frameworks existentes


OmniSecurity

+ ...

Objetivos da JSR-375

"The goal of this JSR is to improve the Java EE platform by ensuring the Security API aspect is useful in the modern **cloud/PaaS application paradigm**."

"This promotes self-contained application portability across **all Java EE servers**, and promotes use of modern programming concepts such as expression language, and contexts and dependency injection."

"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."

JSR 375 - Linha do tempo

Q3 2014 Expert Group formed

NÓS ESTAMOS AQUI (Jul/2015)

Q4 2015 Early Draft

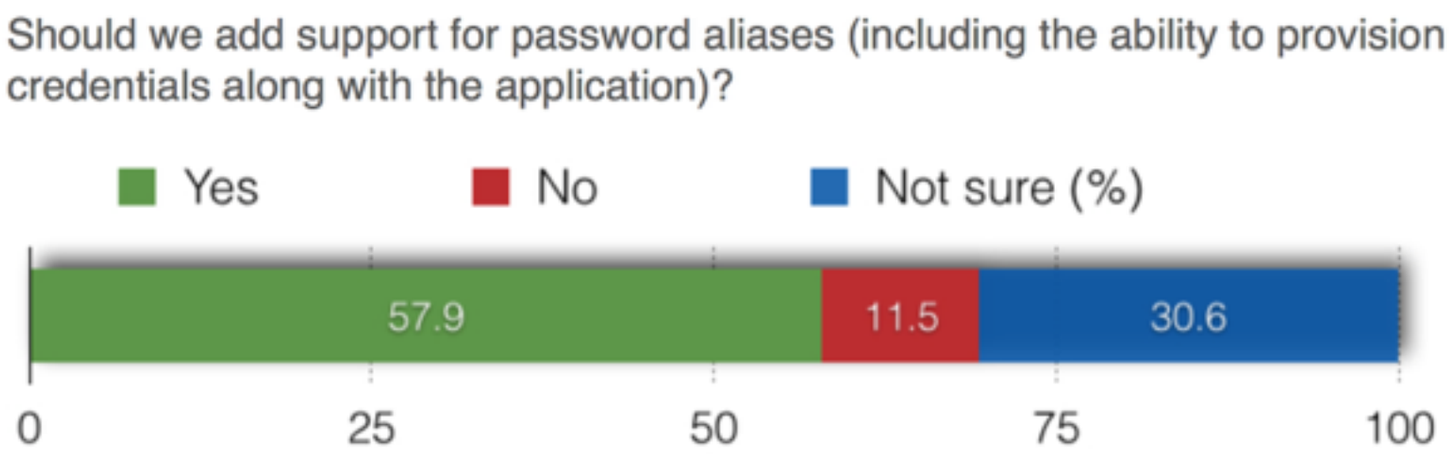
Q1 2016 Public Review

Q3 2016 Proposed Final Draft

H1 2017 Final Release

Quais as prioridades?

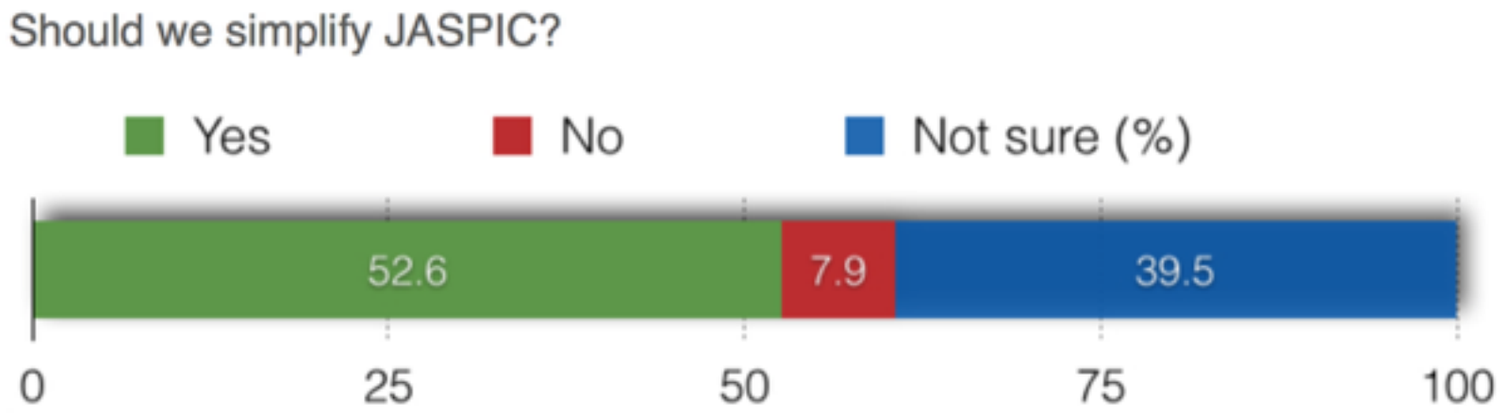
- Baseadas nos resultados finais do Java EE 8 Community Survey
  - **Password aliases**



Fonte:
Java EE 8 Community Survey

Quais as prioridades?

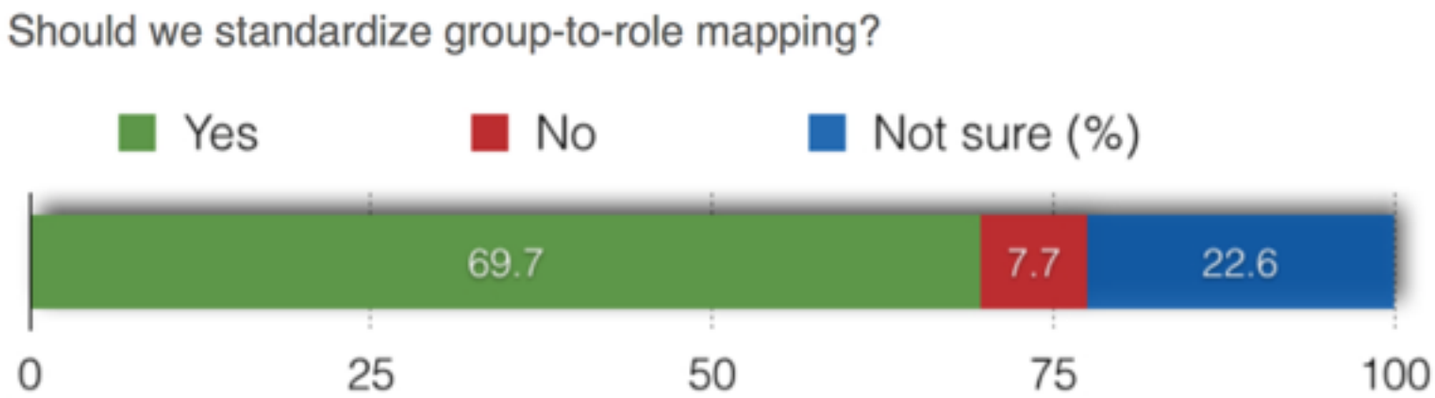
- Baseadas nos resultados finais do Java EE 8 Community Survey
  - Password aliases
  - **API de Autenticação (simplificação do JASPIC)**



Fonte:
Java EE 8 Community Survey

Quais as prioridades?

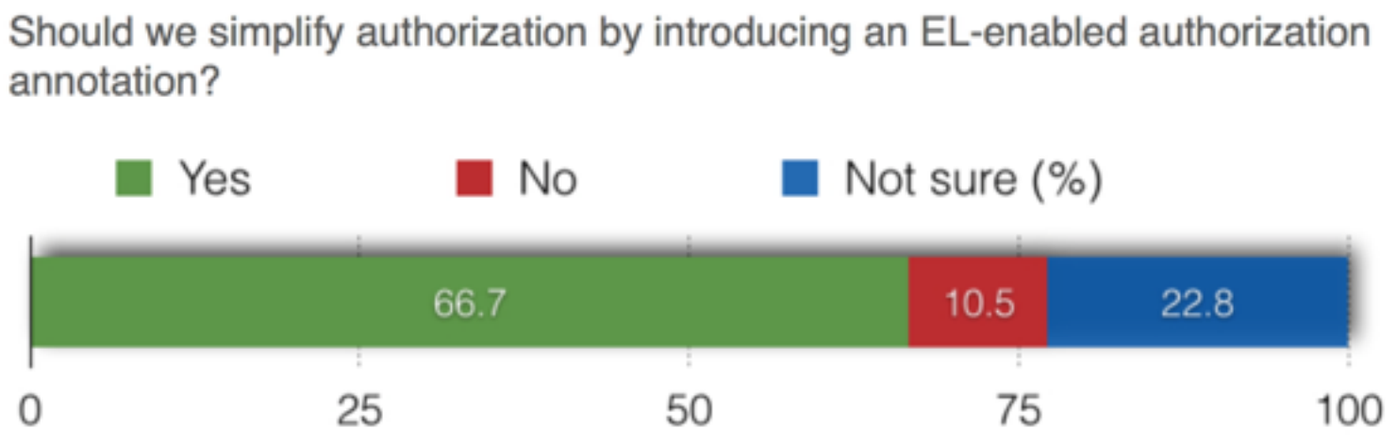
- Baseadas nos resultados finais do Java EE 8 Community Survey
  - Password aliases
  - API de autenticação (simplificação do JASPIC)
  - **API para atribuir roles e permissões**



Fonte:
Java EE 8 Community Survey

Quais as prioridades?

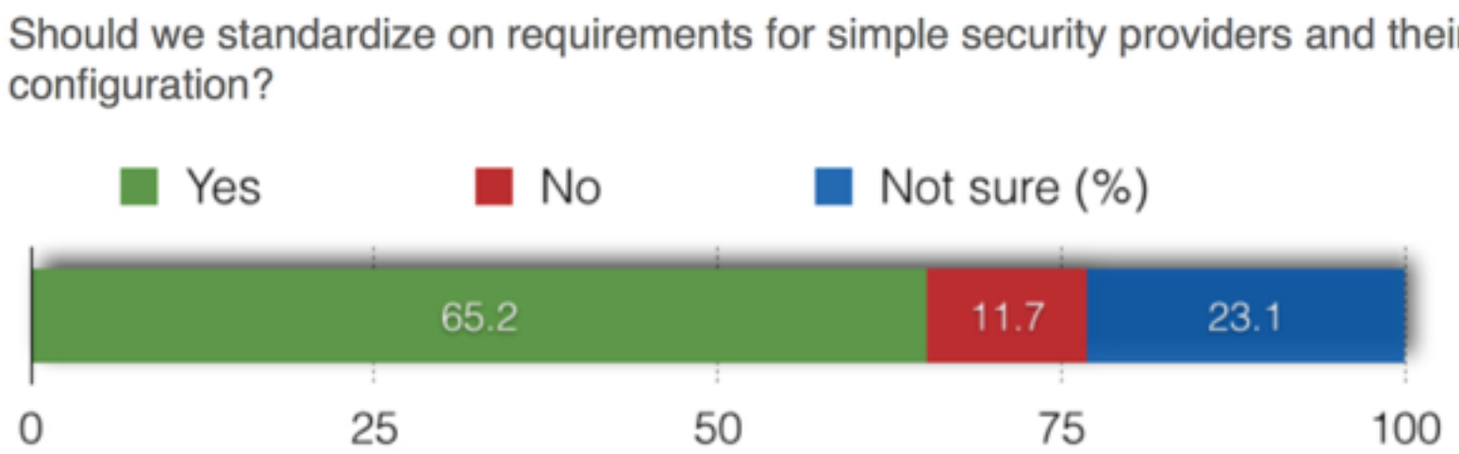
- Baseadas nos resultados finais do Java EE 8 Community Survey
  - Password aliases
  - API de autenticação (simplificação do JASPIC)
  - API para atribuir roles e permissões
  - **Interceptadores de autorização**



Fonte:
Java EE 8 Community Survey

Quais as prioridades?

- Baseadas nos resultados finais do Java EE 8 Community Survey
  - Password aliases
  - API de autenticação (simplificação do JASPIC)
  - API para atribuir roles e permissões
  - Interceptadores de autorização
  - **API para Identity Store**



Fonte:
Java EE 8 Community Survey

Quais as prioridades?

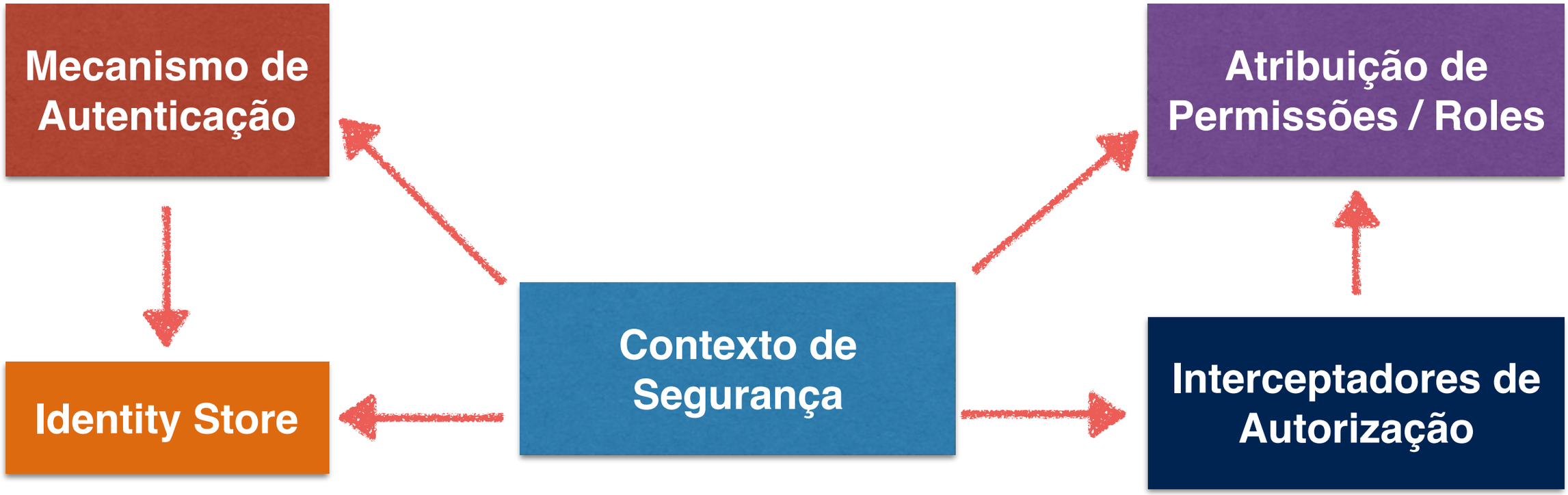
- Resultados finais do Java EE 8 Community Survey
  - Password aliases
  - API de autenticação (simplificação do JASPIC)
  - API para atribuir roles e permissões
  - Interceptadores de autorização
  - API para Identity Store
- **+ Simplificar a API**
  - **Introduzir um contexto de segurança unificado**
  - **Unificar a terminologia**

Principais idéias 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

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Relacionamentos entre as propostas



Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Onde estamos? (24/jul/2015)

https://java.net/projects/javaee-security-spec

Current Stage: *Early Draft Development*

Epics

Name	Status	Links	Description
Terminology	In Progress	epic doc	Establish Security API terminology to enable accurate and concise communication
Authentication Mechanism	In Progress	epic	Simplify application-accessible authentication mechanisms
Identity Store	In Progress	epic examples doc	Standardize application-accessible identity store
Role/Permission Assignment	Not Started		Standardize application-accessible role/permission assignment
Security Context	Not Started		Standardize a platform-wide Security Context
Authorization Interceptors	Not Started		Standardize platform-wide Authorization Interceptors
Password Aliasing	Not Started		Standardize the API for using password aliases in configuration
Standardized Server Authentication Modules	Not Started		Using the simplified Authentication Mechanism, standardize some additional ServerAuthModules

Links:

- epic = Link to JIRA Epic, which is a collection of issues
- examples = Link to code examples
- doc = Link to working document
- spec = Link to specification source text

Terminologia

User
Provider
Rule
X.509
KeyStore
Principal
Module
TrustStore
Grant
Context
Caller
Role
Authority
Hash
Certificate
Authentication
Trust
Realm
Identity
Group
Digest
Token
Domain
Authorization
Key
Signature
Credential
Permission
Subject
Policy
Cypher

Terminologia

- Como se chama a entidade que se autentica?

get????Principal()

get????InRole()

Terminologia

- Como se chama a entidade que se autentica?

getUserPrincipal()

getUserInRole()

Terminologia

- Como se chama a entidade que se autentica?

getCallerPrincipal()

getCallerInRole()

Terminologia

- O que é um **grupo**?

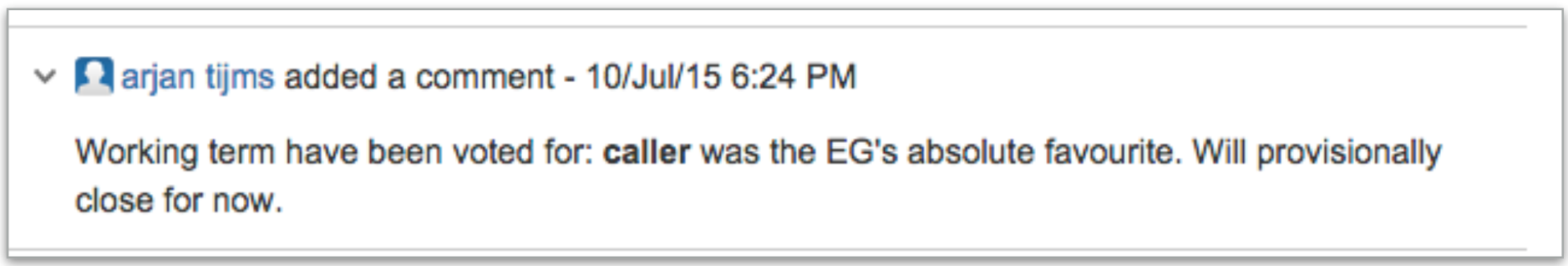
Conjunto de **usuários**?

Uma **permissão**?

??????

Discussão sobre **terminologia**

- Epic (JIRA) sobre terminologia da API
  - **https://java.net/jira/browse/JAVAEE_SECURITY_SPEC-21**
  - \+ em SPECs 1, 2, 3 e 28
- Nas discussões a preferência tem sido maior para "Caller" em vez de "User" para usuário logado (SPEC-2)



Fonte: **https://java.net/jira/browse/JAVAEE_SECURITY_SPEC-2**

Terminologia

authentication repository

provider

- Como se chama a **coisa** que guarda identidades?

identity store

zone

security provider

authenticator

identity manager

login module

domain

realm

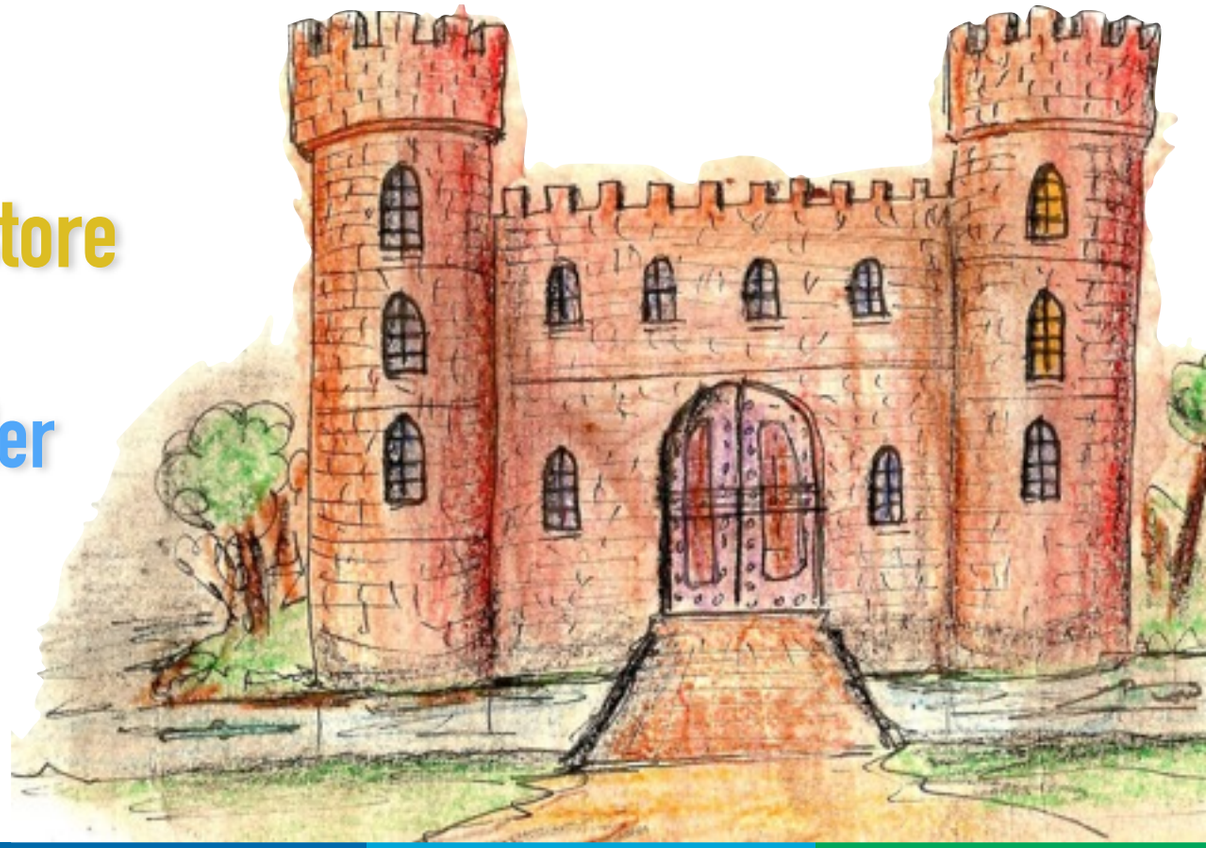
auth store

auth provider

identity provider

region

repository



^63$t+p@55w0rd#

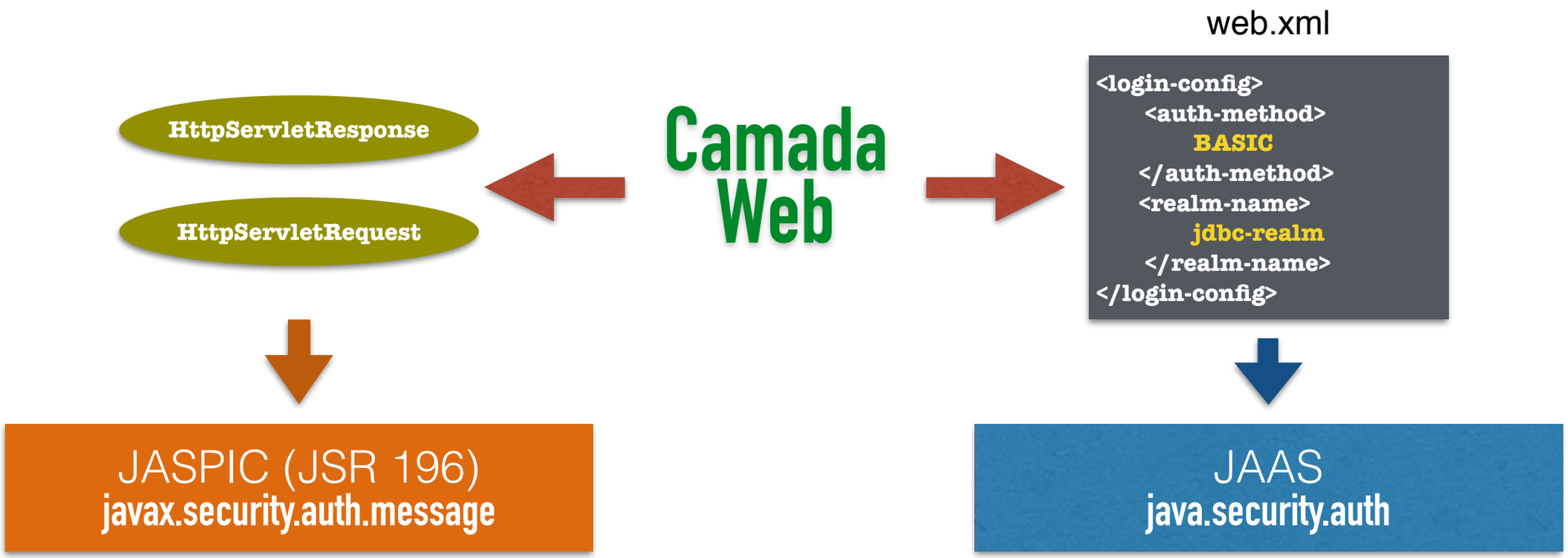
Autenticação

Mecanismo de autenticação

- Situação: desenvolvedor tem uma aplicação que
  - Gerencia seus próprios usuários e grupos
  - Precisa autenticar usuários para atribuir roles
  - Quer autenticar usando modelos do domínio da aplicação
    - Precisa usar método não suportado (ex: OpenID Connect)
- Solução simples
  - Vendor lock-in com o fabricante do servidor de aplicações
  - Framework de terceiros (Spring, Shiro, etc.)
- Solução Java EE: **?**

Autenticação em Java EE

- Java EE oferece duas alternativas (excludentes) de configuração
  - API **declarativa** na **camada Web** para **selecionar realm** e o **método** de autenticação (BASIC, FORM, DIGEST, CLIENT)
  - API programática (**JASPIC**) para criar módulos de autenticação
- A **realização** da autenticação é **dependente de fabricante**
  - Maioria oferece APIs e ferramentas **proprietárias** baseadas em **JAAS**



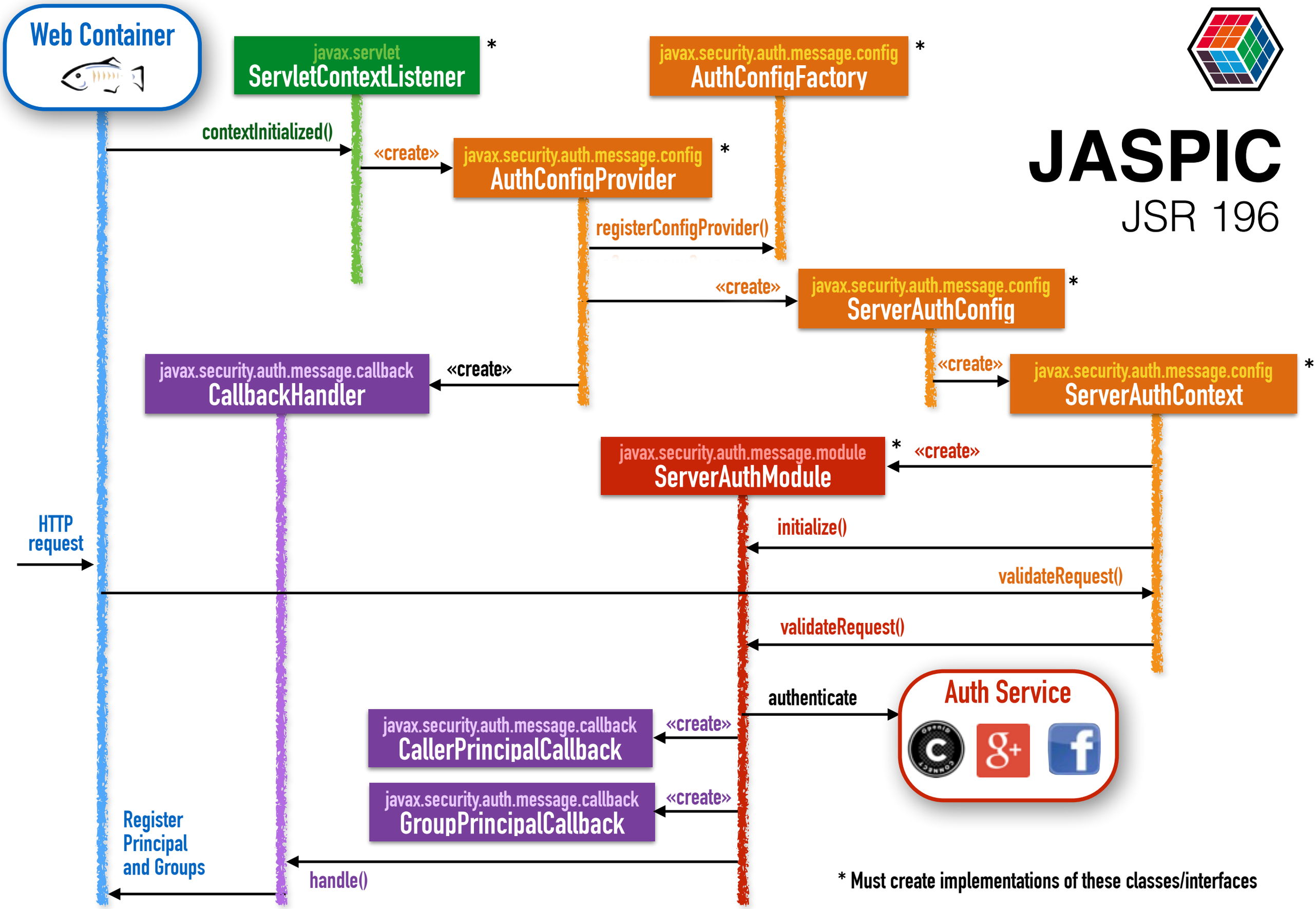
JAAS Principal
!=
Java EE Principal

- O foco é **diferente** em Java SE e Java EE
  - Java SE (**JAAS**): um **Subject** representa um usuário, que pode ter uma coleção de identidades (**Principal**)
  - Java EE: um **Principal** representa um usuário que pode ter uma coleção de permissões (**Roles**)

JASPIC **SAM**

```
public interface ServerAuthModule {

    public void initialize(...) throws AuthException;

    public AuthStatus validateRequest(...);

    public Class<?>[] getSupportedMessageTypes();

    public AuthStatus secureResponse(...);

    public void cleanSubject(...);

}
```

JASPIC
JSR 196



* Must create implementations of these classes/interfaces

Proposta JSR-375: JASPIC

- Por que ninguém usa JASPIC?
  - Complexo! Tem que programar 6 classes
- **Ideia**: reduzir 5 classes a **uma** (SAM)
- Registrar no ServletContextListener e/ou anotações

```
@Authenticator("org.acme.TokenAuthModule")
@WebServlet("/SimpleServlet")
@ServletSecurity(@HttpConstraint(rolesAllowed = {"manager"}))
public class SimpleServlet extends HttpServlet {
```

- Em vez de implementar toda a interface, usar um Profile
  - ex: MySAM extends HttpServerAuthModule {}

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Proposta JSR-375: JASPIC

- Fornecer **autenticadores padrão**

```
@Authenticator("javax.security.authenticator.OpenIDConnect")
@WebServlet("/SimpleServlet")
@ServletSecurity(@HttpConstraint(rolesAllowed = {"manager"}))
public class SimpleServlet extends HttpServlet {
```

- Lançar eventos CDI
  - `@PreAuthenticate`, `@PostAuthenticate`
    `@PreLogout`, `@PostLogout`
  - Para contar usuários logados,
  - Rastrear tentativas de login mal-sucedidas
  - Criar usuário local depois de autenticação remota
  - Preferências do usuário, etc.

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Identity Store

Gerência de usuários: hoje

- Atualmente **não existe suporte padrão** para gerenciar usuários em Java EE
- 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**

Proposta JSR-375: Identity Store

- **Serviço padrão** para criar, atualizar, remover, e agrupar usuários
  - Usa um recurso de usuários
  - Escopo pode ser limitado à aplicação e/ou compartilhado por outras aplicações
  - LDAP, arquivos, datasource, **embedded**, servidor
  - Poder ser ntercambiável: recurso diferente para desenvolvimento, teste, produção
  - API poderia informar quais os serviços disponíveis em determinado recurso de usuários.

Primeiro esboço: API de Identity Store

```
public interface IdentityStore {
    Identity loadIdentityByName(String name);
    void changePassword(char[] oldPassword, char[] newPassword);
    void createIdentity(Identity user);
    void deleteIdentity(String name);
    void updateIdentity(Identity identity);
    boolean identityExists(String name);
    void createGroup(String group);
    void deleteGroup(String group);
    void addIdentityToGroup(String name, String group);
    void removeIdentityFromGroup(String name, String group);
    boolean isIdentityInGroup(String name, String group);
    List<String> getIdentityGroups(String name);
}

public interface Identity {
    String getUsername();
    char[] getPassword();
    boolean isAccountExpired();
    boolean isAccountLocked();
    boolean isPasswordExpired();
    boolean isEnabled();
    IdentityAttributeValue getAttribute(String name);
}
```

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Proposta **+ recente** para Identity Store

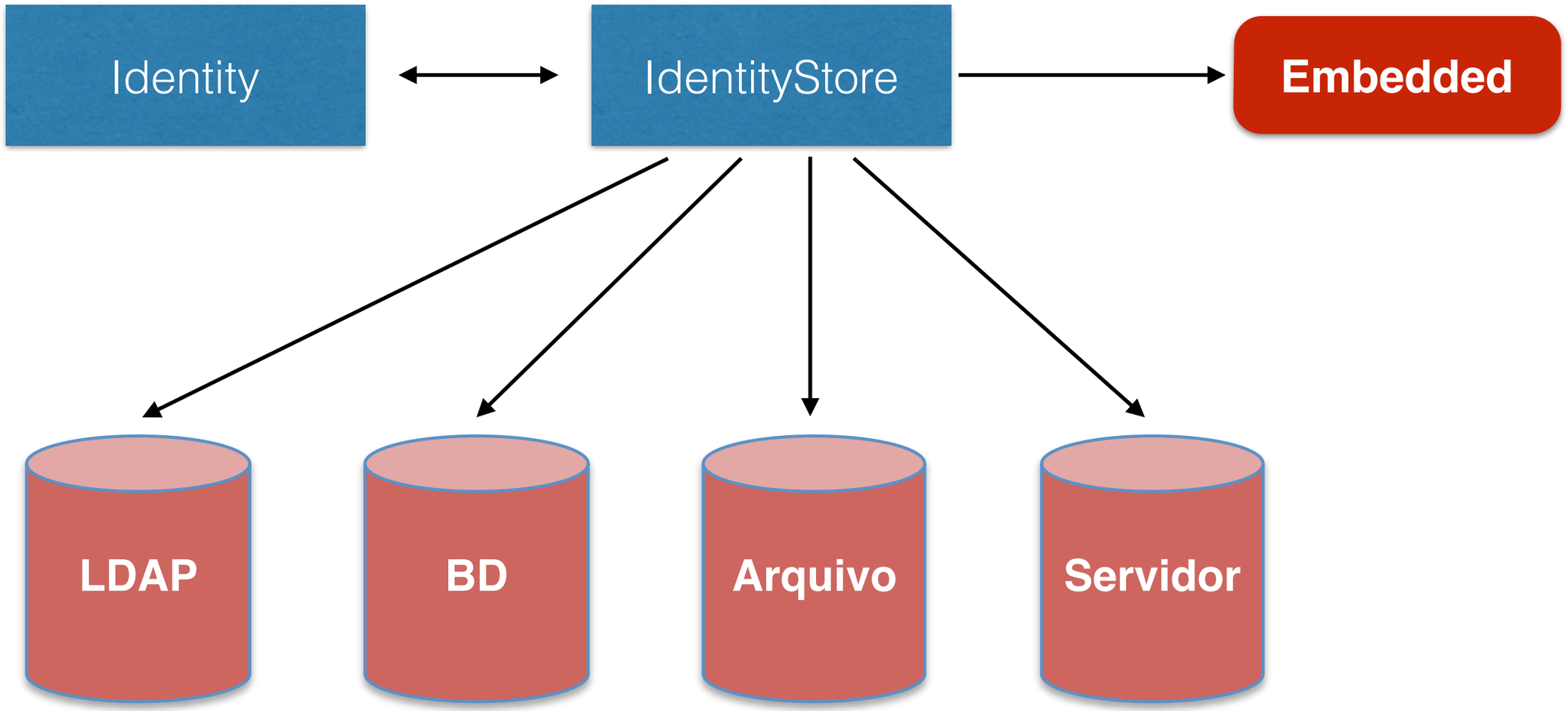
```
public interface IdentityStore {
    // Callers and Groups
    Caller loadCaller(String loginName);
    boolean callerExists(String loginName);
    List<Caller> getCallers(String regEx);
    boolean groupExists(String name);
    List<String> getGroups(String regEx);
    boolean isCallerInGroup(String callerLoginName, String groupName);
    List<Caller> getCallersInGroup(String group);
    List<String> getCallerGroups(String callerLoginName);
    List<String> getCallerGroups(Caller caller);

    // Credentials
    CredentialValidationResult validateCredentials(Credentials credentials);

    // Role Mapping
    boolean roleExists(String name);
    enum RoleSetSelector {ALL, ASSIGNED, UNASSIGNED}
    List<String> getRoles(String regEx, RoleSetSelector roleSet);

    // Roles mapped to Caller //
    boolean callerHasRole(String loginName, String role);
    boolean callerHasRole(Caller caller, String role);
    List<Caller> getCallersWithRole(String role);
    List<String> getRolesForCaller(String loginName);
    List<String> getRolesForCaller(Caller caller);

    // Roles mapped to Group //
    boolean groupHasRole(String groupName, String role);
    List<String> getGroupsWithRole(String role);
    List<String> getRolesForGroup(String groupName);
}
```

Fonte:
https://github.com/javaee-security-spec/javaee-security-proposals
Jun 08 2015

Baseada em **PicketLink**
(picketlink.org)

```
public class Caller {
   private String loginName;

   public String getLoginName() {
      return loginName;
   }

   public void setLoginName(String loginName) {
      this.loginName = loginName;
   }
}
```

Identity Store: implementação



Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Identity store: **idéias** de como usar

- Como embutir um Identity Store na própria aplicação

```
@EmbeddedIdentityStore(name="java:app/devIdentityStore",
{
    @Identity(username="ray", password="secret", groups="admin"),
    @Identity(username="jo", password="secret", groups="user"),
    @Identity(username="sam", password="secret", groups="user")
})
```

- Identity Store em banco de dados relacional

```
@DatabaseIdentityStore(
    name="java:app/testIdentityStore",
    lookup="somedatabase",
    userQuery="SELECT passwd FROM principals WHERE userid=?",
    groupQuery="SELECT group FROM groups where username=?",
 ...)
```

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Password aliasing

58:87:52:44:D8:60:12:B0:
FB:D5:F6:C0:6E:F1:6E:FC:
A2:0E:15:8D:58:E9:6E:6F:
76:CE:DA:66:60:B5:9B:C2

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA950rM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

Password Aliasing

- Atualmente **não há suporte padrão** para referenciar e guardar uma senha (para acessar recursos) de forma segura em Java EE
  - Aplicações precisam de uma senha assim, por exemplo, para acessar recursos de bancos de dados, em um **persistence.xml**
- Soluções são in-house ou usando ferramentas de terceiros

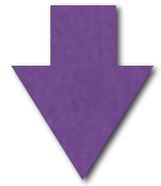
Proposta JSR-375: Password Aliasing

- **Sintaxe padrão** para indicar um alias de senha, e um meio para resolver o alias para um valor de senha
  - Repositório seguro que poderia ser mantido com a aplicação

```
@DataSourceDefinition(
    name="java:app/jdbc/test",
    user="root",
    password="${ALIAS=token}", ...)
```

```
<data-source>
<name>java:app/env/testDS</name>
<user>APP</user>
<password>${ALIAS=token}</password>
...
</data-source>
```

${ALIAS=a#6W@+2&3_}



"senha"

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Autorização e role-mapping

Autorização em Java EE

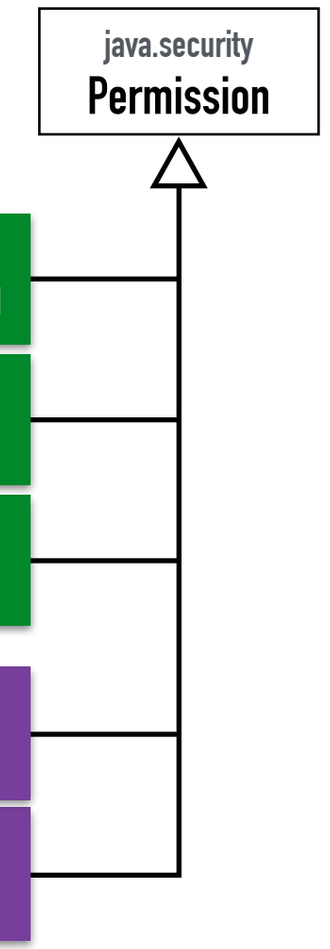
- Baseada em **roles**, fácil de usar e 100% Java EE*
  - Configuração **declarativa** via anotações e deployment descriptor XML, ou através de API programática (**JACC**)
  - Acesso (leitura) através APIs programáticas em **contextos de segurança**

	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)



* depois de mapeados grupos/usuários a roles

Role Mapping em Java EE

- Atualmente **não há suporte padrão** a mapeamento de roles em Java EE
- Soluções implicam em **vendor lock-in** ou uso de APIs de terceiros

```
<glassfish-web-app>
    <context-root>JavaEESecurity</context-root>

    <security-role-mapping>
        <role-name>administrador</role-name>
        <principal-name>cerebro</principal-name>
        <group-name>alienigenas</group-name>
    </security-role-mapping>

    <security-role-mapping>
        <role-name>especial</role-name>
        <principal-name>vini</principal-name>
        <principal-name>masha</principal-name>
    </security-role-mapping>

    <security-role-mapping>
        <role-name>amigo</role-name>
        <principal-name>masha</principal-name>
        <principal-name>vini</principal-name>
        <principal-name>niquel</principal-name>
    </security-role-mapping>

    <security-role-mapping>
        <role-name>outro</role-name>
        <principal-name>kusko</principal-name>
    </security-role-mapping>
</glassfish-web-app>
```

Proposta JSR-375: Role Mapping

- **Serviço de roles padronizado** que permite que a aplicação realize operações de mapeamento de roles
  - granting, pesquisa, para roles de usuários e grupos
  - Serviço manipularia mapeamentos de um role mapper (pode ser **embedded**, JDBC, LDAP, etc.)

```
@EmbeddedRoleMapper(
   users={
      @RoleMap(user="foo",roles="admin"),
      @RoleMap(group="admin",roles={"admin","manager"})
   }
)
public class MyServlet {
}
```

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Role Mapper **dinâmico**

```
public interface RoleMapper {
    void grantRoleToUser(String username, String role);
    void revokeRoleFromUser(String username, String role);
    boolean hasRoleForUser(String username, String role, boolean includeGroups);
    List<String> getRolesForUser(String username, boolean includeGroups);
    List<String> getUsersWithRole(String role, boolean includeGroups);
    void grantRoleToGroup(String group, String role);
    void revokeRoleFromGroup(String group, String role);
    boolean hasRoleForGroup(String group, String role);
    List<String> getRolesForGroup(String group);
    List<String> getGroupsWithRole(String role);
}
```

```
@Inject RoleMapper roleMapper;
roleMapper.grantRoleToGroup("Manager", "EDIT_ACCOUNTS");
boolean has = roleMapper.hasRoleForGroup("Manager", "CLOSE_ACCT");
List<String> roles = roleMapper.getRolesForGroup("Manager");
List<String> groups = roleMapper.getGroupsWithRole("VIEW_ACCOUNTS");
```

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Limitações da autorização

- Atualmente Java EE apenas suporta **autorização por permissões** (roles)
- Não há suporte padrão para incluir **regras** do domínio da aplicação na decisão de autorização
- Soluções são in-house ou proprietárias

Proposta JSR-375: interceptadores

- Anotação padrão de **interceptador** para métodos, capaz de incluir regras que seriam usadas na autorização
  - Seria chamada se forma **similar** a @AroundInvoke
  - As regras teriam acesso ao contexto atual
  - Regras poderiam ser baseadas em texto (como **Java EL**)
  - Suporte a **eventos CDI** como parte do processo de decisão

```
@LdapAuthorizationRules (
      name="java:app/accountAuthRules",
      ldapUrl="ldap://blah",
      ldapUser="ElDap",
      ldapPassword="mysecret"
)
public class MyBean {
   @ EvaluateSecured(ruleSourceName="java:app/accountAuthRules",
                     rule="transferFunds")
   void transferFunds() {..};
}
```

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

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

JSF backing beans

getUserPrincipal()

isUserInRole()

javax.faces.context.
ExternalContext

CDI

@Inject
java.security.Principal

WebSockets

getUserPrincipal()

javax.websocket.
Session

RESTful Web Services

getUserPrincipal()

isUserInRole()

javax.ws.rs.core.
SecurityContext

Proposta JSR-375: contexto de segurança

- Deveria haver apenas **um** contexto de segurança que pudesse ser obtido por **qualquer** componente Java EE

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

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

Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Exemplo (idéia)

```
public interface SecurityContext {
    String getUserPrincipal();
    boolean isUserInRole(String role);
    List<String> getAllUsersRoles();
    boolean isAuthenticated();
    boolean isUserInAnyRole(List<String> roles);
    boolean isUserInAllRoles(List<String> roles);
    void login(Object request, Object response);
    void login(Map map);
    void logout();
    void runAs(String role);
    boolean hasAccessToResource();
    boolean hasAccessToBeanMethod();
}
```

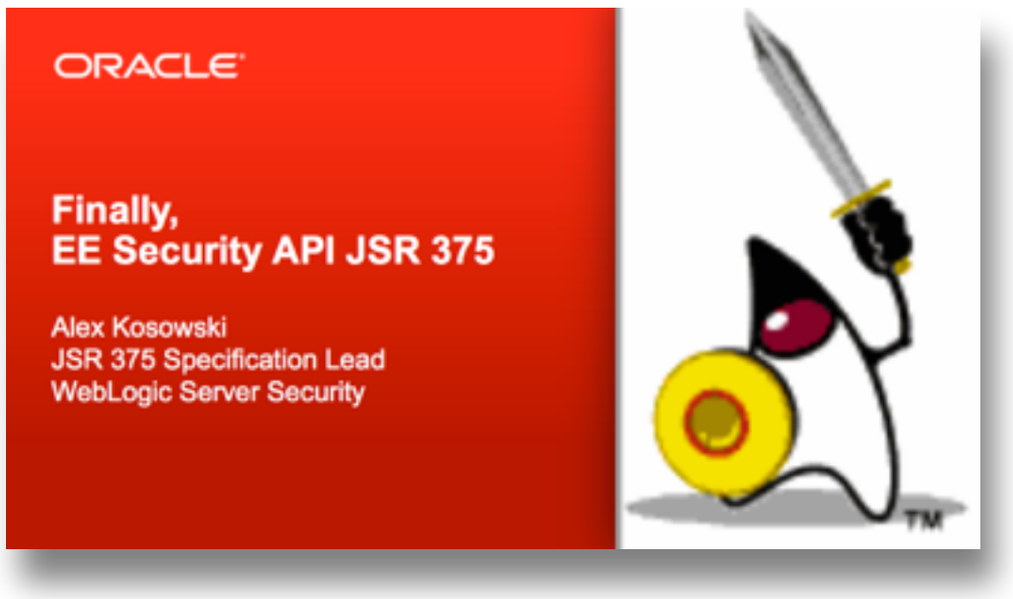
Fonte:
Palestra de Alex Kowolski
Finally, Security API JSR 375
Devoxx France Apr 08 2015

Conclusões

- Java EE **não tem** uma API de segurança padrão independente de plataforma, moderna e fácil de usar
- Java EE 8 **pode ser que tenha** - essa é a meta do JSR-375
- Se você tem interesse, **envolva-se**!
- **Como se envolver**
  - Página do **projeto**: https://java.net/projects/javaee-security-spec
  - **Lista** de discussões: users@javaee-security-spec.java.net
  - **Playground** no GitHub (faça **fork** e explore): https://github.com/javaee-security-spec/javaee-security-proposals

Referências

- Página da JSR-375
  - https://www.jcp.org/en/jsr/detail?id=375
- Discussões do **JIRA** do JSR-375 e **listas de e-mail**
  - https://java.net/jira/browse/JAVAEE_SECURITY_SPEC
  - https://java.net/projects/javaee-security-spec/lists
- Artigos de Arjan Tijms sobre JASPIC e JAAC
  - arjan-tijms.omnifaces.org
- Palestra do **spec lead** Alex Kosowski no DevoXX France (Abril 2015)



obrigado!

helder@argonavis.com.br

Palestra

http://www.argonavis.com.br/download/tdc2015_javaee8_security.html

Código-fonte (Experimentos com JASPIC e JACC)

https://github.com/argonavisbr/JavaEE7SecurityExamples