Módulo 7

Segurança

Lição 6

JAAS

Versão 1.0 - Jan/2008

Autor
Aldwin Lee
Cheryl Lorica

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Angelo de Oliveira
Bruno da Silva Bonfim
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Guilherme da Silveira Elias
Leandro Souza de Jesus
Lucas Vinícius Bibiano Thomé
Luiz Fernandes de Oliveira Junior
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach e Vinícius G. Ribeiro (Especialista em Segurança)
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

A tecnologia JAAS – Java Authenthication and Autorization Service (Serviço de Autorização e Autenticação Java) fornece uma interface para autenticar usuários e conceder autorização baseada na identidade do usuário ao invés das características da fonte. Com JAAS é possível restringir um aplicativo baseado na sua localização, no seu autor (quem o assinou) ou no seu executor.

JAAS disponibiliza um *framework* através de uma arquitetura *plug-in-play*. Fornece um conjunto de classes abstratas e, no momento de sua execução, procura por um *provider* necessário à classe. No entanto, esta arquitetura não está acoplada ao *framework* de segurança. Por isso iremos nos restringir em consultar as principais classes de JAAS bem como seus detalhes de funcionamento.

Ao final desta lição, o estudante será capaz de:

- Trabalhar com a tecnologia JAAS
- Identificar as políticas de segurança de JAAS e seus arquivos de configuração
- Conhecer as classes de autenticação e autorização da tecnologia JAAS
- Programar e administrar através da tecnologia JAAS

2. Trabalhando com JAAS

Uma aplicação que possui JAAS habilitada trabalha da seguinte forma:

1. O *framework* solicita ao usuário que efetue *login*, obtendo assim um objeto *login* de usuário.

Esta é uma operação simples para um programador, uma vez que envolve apenas a construção de um objeto do tipo *LoginContext* e a invocação de um único método. A complexidade da tarefa vem após a invocação do método implicando, assim, em um esforço junto ao administrador do sistema.

O administrador do sistema é responsável pela criação de um arquivo que contém uma ou mais diretivas que indicam o que acontece quando uma determinada aplicação possui um arquivo de registro com várias tentativas de *login*. Essas diretivas estão em forma de módulos de *login*, que são invocados para autenticar determinado usuário e uma série de opções que indicam como essas classes podem ser utilizadas. As próprias classes tipicamente interagem com o sistema operacional que, por sua vez, pede para autenticar o usuário utilizando os sistemas de autenticação disponíveis na plataforma.

O administrador do sistema também determina os parâmetros de autenticação. Alternativamente poderá obrigar a informar uma senha válida, seja uma LDAP (utilizada em sistema *Solaris*), NT (utilizada em sistema *Windows*) ou outra, utilizada por uma base de dados customizada. O uso de uma ou mais senhas é opcional. O administrador pode trabalhar com poucos ou com muitos módulos de *login*, conforme se fizer necessário.

2. O programa chama o método *doAs*() ou *doAsPrivileged*() recebendo do objeto *login* o código que pode ser executado com a permissão desse usuário.

Este passo é idêntico à chamada ao método *doPrivileged*() na lista do controlador de acesso.

3. Dentro do contexto criado, apenas o programa executa o código que requer uma permissão específica (ex.: mostrar os arquivos em um diretório).

Assim como em todas as solicitações, esse código resulta em uma chamada ao gerenciador de segurança (conhecido como Controlador de Acesso) para verificar se o objeto possui a permissão apropriada. Normalmente são concedidas as devidas permissões a todas as classes conforme o arquivo de politica de segurança padrão (ou qualquer que seja a classe de política de segurança que esteja em vigor).

2.1. Arquitetura JAAS

A arquitetura JAAS é dividida em dois componentes principais: o componente de autenticação e o componente de autorização.

O componente de autenticação fornece segurança e confiabilidade para determinar quem está executando o código Java, independentemente se o código está sendo executado através de uma aplicação, uma *Applet*, um *JavaBean* ou uma *Servlet*.

O componente de autorização fornece recursos para restringir a ação de uma classe Java quanto a execução de tarefas sensíveis baseadas na fonte na qual originou esta execução (ou seja, de onde veio esta classe) e como foi autenticada.

2.2. As classes comuns de JAAS

Aqui estão as principais classes e interfaces do JAAS que podem ser usadas, estendidas ou implementadas para tratar autorização e autenticação.

- javax.security.auth.AuthPermission
- javax.security.auth.Policy
- javax.security.auth.Subject
- javax.security.auth.PrivateCredentialPermission

- javax.security.auth.login.Configuration
- javax.security.auth.login.LoginContext
- javax.security.auth.spi.LoginModule (Interface)
- javax.security.auth.callback.Callback (Interface)
- javax.security.auth.callback.CallbackHandler (Interface)
- javax.security.auth.callback.ChoiceCallback
- javax.security.auth.callback.ConfirmationCallback
- javax.security.auth.callback.LanguageCallback
- javax.security.auth.callback.NameCallback
- javax.security.auth.callback.PasswordCallback
- javax.security.auth.callback.TextInputCallback
- javax.security.auth.callback.TextOutputCallback

3. Políticas de segurança JAAS e arquivos de configuração

Os seguintes arquivos são utilizados pela JAAS para modificar as autorizações e autenticações conforme o comportamento do sistema:

1. Arquivo para configuração de autenticação

Este é um arquivo de configuração, indicando o objeto do tipo *LoginModule* que será utilizado pela aplicação de forma adaptável e com uma estrutura em formato de *stack*. Esse arquivo pode ser modificado para informar uma *LoginModule* diferente e que possa ser utilizado por uma aplicação sem que seja necessário modificá-la.

Os módulos de *Login* são adaptáveis pois podem ser carregados dinamicamente. Ao invés de chamar um módulo específico de *Login* inserido no código, o contexto de *Login* procura o arquivo de configuração, para verificar quais classes devem ser chamadas. Isto permite a utilização de módulos de *Login* de terceiros.

Os módulos de *Login* estão em uma estrutura em formato de *stack* pois é possível especificar mais de um módulo no arquivo de configuração. Estes módulos são postos em uma *stack* dentro do arquivo de configuração. São chamados em uma determinada seqüência e cada um pode adicionar um ou mais objetos principais para determinado assunto (por exemplo, conhecer o usuário atual). Assim, trata-se do assunto dos objetos que inserem com múltiplos objetos principais, que podem vir de um único módulo ou podem vir de vários módulos de *Login*.

A sintaxe do arquivo de configuração está representeada abaixo:

```
ApplicationName {
  LoginModule flag ModuleOptions;
  <mais entradas no módulo LoginModule>
};
```

Há quatro valores para a *flag*:

- ***required*** – Este módulo será sempre chamado e o usuário deve passar neste teste de autenticação
- ***sufficient*** – Se o usuário passar no teste de autenticação deste módulo, nenhum outro módulo (com exceção do *required*) será executado e o usuário estará suficientemente autenticado
- ***requisite*** – Se o usuário passar no teste de autenticação deste módulo, outros módulos poderão ser executados, porém (exceto o *required*) poderão falhar
- ***optional*** – É permitido ao usuário falhar na autenticação deste módulo. No entanto se todos os módulos forem *optional*, o usuário deverá passar pelo menos no teste de autenticação

Os módulos *optional* são usualmente definidos localmente dentro de um módulo *LoginModule*, podendo, assim, enviar o valor destas opções ao módulo de *Login* através do arquivo de configuração.

A seguir, vemos um arquivo de configuração de *Login*:

```
Application A {
  com.sun.security.auth.module.SolarisLoginModule required;
  com.sun.security.auth.module.JndiLoginModule optional;
};
Application B {
  com.sun.security.auth.module.NTLoginModule required;
};
```

2. Arquivo de configuração de políticas de segurança

Um arquivo de política de segurança JAAS é muito similar a um arquivo de politicas de segurança

padrão. A sintaxe é quase a mesma e os tipos de permissões são exatamente iguais. A única diferença é a possibilidade de especificar o tipo e o nome principal de cada entrada. Isto significa que a entrada informada especifica os *codebases* adicionais, além dos principais, e o código das assinaturas que são utilizados no arquivo de configuração de politicas de segurança.

As entradas neste arquivo aplicam-se a todo código executado pelo método *doAs*() analisado anteriormente; mapas deste arquivo principal são associados ao contexto chamando o método *doAs*() específico para as permissões.

Cada entrada de concessão inclui uma ou mais "permissões de entrada" e precedida pelas opções *codeBase* ou *signedBy* contendo pares de nome e valor que especificam que este código pode conceder a permissão. Temos a seguir, o formato básico de uma entrada de concessão:

```
grant signedBy "signer_names", codeBase "URL" {
  permission permission_class_name "target_name", "action",
    signedBy "signer_names";
  ....
  permission permission_class_name "target_name", "action",
    signedBy "signer_names";
};
```

O *signedBy* e *codeBase* são pares opcionais contendo nome e valor, e a ordem entre estes campos não é relevante. Um valor *signedBy* indica o apelido para um certificado fornecido na *database* de chaves. Múltiplos apelidos podem ser separados por vírgulas. Se for omitido isso significa nenhuma assinatura. Não importa qual dos dois códigos é assinado ou não ou por quem.

Um valor de *codeBase* indica o local do código fonte; permissões serão concedidas com base nesse local. Um *codeBase* vazio significa ausência de local; não importa onde o código se origina.

Veja uma amostra uma entrada de concessão do JAAS:

```
grant
  codebase "file:/files/sdo/jaas/actions/"
  signedBy "jra"
  Principal com.sun.security.auth.SolarisPrincipal "sdo" {
    permission java.io.FilePermission "${/}files", "read";
};
```

4. JAAS – Classes de Autenticação

Vamos discutir em maiores detalhes a classe e a interface necessárias para permitir que JAAS execute autenticação.

4.1. Classe LoginContext

```
javax.security.auth.login.LoginContext
```

Esta classe é utilizada para descrever os métodos básicos usados para autenticação e permite desenvolver um aplicativo independente da tecnologia de autenticação envolvida. *LoginContext* consulta um arquivo de configuração para determinar quais *LoginModule* devem ser usados. Estes são os métodos vitais para esta classe:

```
public LoginContext(String name)
public LoginContext(String name, CallbackHandler cb)
public LoginContext(String name, Subject s)
public LoginContext(String name, Subject s, CallbackHandler cb)
```

Estabelecer um contexto pelo qual um usuário pode ser autenticado. As ações que esta autenticação irá executar são carregadas através de um objeto de configuração. O argumento *name* é a identificação utilizada por esse conjunto de ações dentro do objeto de configuração.

```
public void login()
```

Executar a autenticação e realiza as ações listadas na configuração. Este método lança *LoginException* se o *login* falhar.

```
public void logout()
```

Informar que o usuário está saindo do sistema, isso invalida o objeto *subject*. Este método lança *LoginException* se a operação de logout falhar.

```
public Subject getSubject()
```

Retorna o objeto do tipo Subject que representa o usuário.

4.2. Interface LoginModule

```
javax.security.auth.spi.LoginModule
```

Esta interface é implementada por serviços de autenticação. Objetos *LoginModule* dão suporte às aplicações para fornecer um determinado tipo de autenticação. Estes são os métodos necessários para implementar a interface.

```
public void initialize(Subject s, CallbackHandler ch, Map sharedState, Map
options)
```

Inicializar o *LoginModule*. O objeto do tipo *Subject* representa o usuário que será autenticado; o módulo de *login* irá armazenar um ou mais usuários, e talvez utilizará o objeto do tipo *CallbackHandler* para obter informações de autenticação diretamente com o usuário. O primeiro objeto do tipo *Map* serve para compartilhar informações. O segundo objeto tipo *Map* contém as opções carregadas do arquivo de configuração.

```
public boolean login()
```

Autenticar o usuário. Informações sobre o usuário podem ser obtidas no ambiente ou utilizando o *CallbackHandler*. Este método retorna *true* caso o usuário obtenha sucesso na autenticação ou *false* se a autenticação deve ser ignorada. Se, o usuário não puder ser autenticado este método deve lançar um *LoginException*.

```
public boolean commit()
```

Confirmar o processo de autenticação. Este método é chamado somente se o usuário for

autenticado para todos os *LoginModule* do arquivo de configuração. Neste ponto, o *LoginModule* deve vincular os objetos *Principal* apropriados ao usuário. Se, por algum motivo, os *LoginModule* desse usuário devam ser ignorados, este método deve retornar *false*. Um *LoginException* é lançado em caso de problema com a confirmação da autenticação.

```
public boolean abort()
```

Interromper o processo de autenticação. Este método é chamado quando o usuário não pode ser autenticado, isto é, um módulo exigido falhou ou não conseguiu obter um módulo adicional. O módulo deve limpar qualquer estado armazenado. Pode lançar um *LoginException* ao encontrar um erro.

```
public boolean logout()
```

Realizar a saída do usuário. Implica na limpeza de qualquer estado e remoção dos objetos do tipo *Subject* e qualquer assunto salvos.

4.3. Classe Subject

```
javax.security.auth.Subject
```

Esta classe é usada para representar um usuário autenticado. Na sua essência, cada usuário é representado como um *array* de objetos Principal guardados por esta classe. Existe um *array* de objetos, pois cada usuário terá, muito provavelmente, várias características de identificação.

4.4. Interface Callback

```
javax.security.auth.callback.CallBack
```

Implementações da interface *CallBack* são passadas a um *CallbackHandler*, permitindo que serviços de segurança possam interagir com uma chamada de aplicação para recuperar dados de específicos de autenticação específica ou para exibir algumas informações.

4.5. Implementações da interface Callback

JAAS traz algumas implementações da interface *Callback*.

```
javax.security.auth.callback.ChoiceCallback
```

Usada para exibir uma lista de opções e recuperar as opções selecionadas.

```
javax.security.auth.callback.ConfirmationCallback
```

Usada para pedir confirmações do tipo sim, não e cancelar.

```
javax.security.auth.callback.LanguageCallback
```

Usada para recuperar o local utilizado para formatação de texto.

```
javax.security.auth.callback.NameCallback
```

Utilizada para obter a informação do nome.

```
javax.security.auth.callback.PasswordCallback
```

Usada para obter informações de senha.

```
javax.security.auth.callback.TextInputCallback
```

Usada para obter informações de texto.

```
javax.security.auth.callback.TextOutputCallback
```

Usada para mostrar mensagens informativas, de avisos e erros.

```
javax.security.auth.callback.CallbackHandler
```

Ao se construir um *LoginContext*, tem-se a opção de fornecer um objeto que implementa esta interface. Este objeto é enviado para os módulos de *Login* e, caso necessitar de uma informação do usuário, usam o objeto *handler* para obtê-la. A implementação de uma *CallbackHandler* exige um objeto que forneça este método:

```
public void handle (Callback [] cb)
```

Recebe um *array* de objetos tipo *Callback* e procura a informação desejada em cada um deles. A aplicação tem liberdade para usar qualquer método para obter a informação adequada. Se a aplicação não sabe como lidar com um determinado objeto do tipo *Callback*, deve lançar uma *UnsupportedCallbackException*; outros erros podem ser encapsulados como uma *IOException*.

5. JAAS – Classes de Autorização

Estas são as classes de autorização utilizadas pelo JAAS.

5.1. Classe Policy

```
java.security.Policy
```

A implementação do arquivo de políticas de segurança do JAAS é fornecido por esta classe. É semelhante a construção da classe *javax.security.auth.Policy*, mas está ligada ao núcleo de políticas de segurança

O classe *Policy* possui quatro métodos principais:

```
public static Policy getPolicy()
```

Retornar o objeto da classe *Policy* instalado. Deve ter a *AuthPermission* denominada *getPolicy* para chamar este método. O retorno deste método é um objeto que contém as políticas de segurança do JAAS em vigor.

```
public static void setPolicy(Policy policy)
```

Determinar as políticas de segurança do JAAS. Deve ter a *AuthPermission* denominada *setPolicy* para poder chamar este método.

```
public abstract PermissionCollection getPermissions(Subject subject,
CodeSource cs)
```

Obter as permissões que devem ser concedidas. São carregadas a partir do código fonte quando executado pelo objeto tipo *Subject* informado.

```
public abstract void refresh()
```

Atualizar as políticas de segurança em vigor.

5.2. Classe AuthPermission

```
javax.security.auth.AuthPermission
```

Esta classe estende a classe *BasicPermission* e pode ser usada no arquivo de políticas de segurança geral da tecnologia Java para especificar os tipos de *AuthPermission* que podem ser concedidos. O construtor para esta classe é:

```
public AuthPermission(String name)
```

A classe concede permissão com base no parâmetro especificado no construtor.

5.3. Classe PrivateCredentialPermission

```
javax.security.auth.PrivateCredentialPermission
```

Esta classe é usada para proteger o acesso às credenciais (objetos do tipo *Credential*) particulares pertencentes a um determinado *Subject*. Esta classe estende da classe *java.security.Permission*.

```
public PrivateCredentialPermission (String name, String actions)
```

Construtor da classe. O argumento *name* indica a classe *Credential* e o conjunto de *Principals*. O único valor válido para o parâmetro *actions* é "read" (leitura).

6. JAAS – Programação e Administração

Agora que os conceitos básicos já foram passados, iremos configurar uma aplicação com autenticação e autorização manipuladas pelo JAAS:

6.1. Particionar o código de configuração e o código de ação e, então, compilá-los

Há três passos importantes na codificação de uma aplicação com JAAS ativado:

1. Construir um objeto *LoginContext*
2. Utilizar este objeto para registrar um usuário
3. Passar este usuário como um dos argumentos para o método *doAs*().

Veja a seguir, um exemplo com autenticação (e autorização) JAAS ativada:

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

public class CountFiles {
  static LoginContext lc = null;

  static class NullCallbackHandler implements CallbackHandler {
    public void handle (Callback [] cb) {
      throw new IllegalArgumentException ("Não implementado ainda");
    }
  }
  public static void main (String [] args) {
    // Usa o LoginModules configurado para a entrada "CountFiles"
    try{
      lc = new LoginContext("CountFiles", new NullCallbackHandler());
    } catch (LoginException le) {
      le.printStackTrace ();
      System.exit(-1);
    }
    // Autentica o usuário
    try{
      lc.login ();
      // Se não for retornada uma exceção, a autenticação foi realizada
    } catch (Exception e) {
      System.out.println ("Login falhou:" + e);
      System.exit(-1);
    }
    // Agora executa o código como o usuário autenticado
    Object o = Subject.doAs (lc.getSubject(), new CountFilesAction());
    System.out.println ("Usuário " + lc.getSubject () +
          " encontrou " + o + " arquivos.");
    System.exit (0);
  }
}
```

No exemplo acima, cria-se um objeto de *LoginContext* usado para estabelecer um contexto pelo qual um usuário pode ser autenticado. O usuário não é autenticado ao se criar o *LoginContext*, isto é feito através do método *login*() em nosso exemplo. Considerando que a autenticação foi bem sucedida, a classe *Subject* é usada para representar um usuário autenticado. A chamada ao método *doAs*() irá chamar o método *run*() do objeto dado, a partir de um determinado *subject* (*CountFilesAction*).

A classe *CountFilesAction* foi definida como:

```
import java.io.*;
import java.security.*;
```

```
class CountFilesAction implements PrivilegedAction {
  public Object run () {
    File f = new File (File.separatorChar + "files");
    File fArray [] = f.listFiles ();
    return new Integer (fArray.length);
  }
}
```

6.2. Criar o arquivo de configuração da autenticação

Após realizarmos a programação, iremos proceder a parte da administração. Definiremos o arquivo de configuração da autenticação do nosso exemplo de programa de JAAS ativado:

```
CountFiles {
      com.sun.security.auth.module.SolarisLoginModule required;
      com.sun.security.auth.module.JndiLoginModule optional;
};
```

A entrada *CountFiles* no primeiro exemplo é comparada com o nome que é passado para o construtor de contexto do *Login*. Quando uma entrada é encontrada, cada uma das classes listadas são chamadas na ordem.

6.3. Criar o arquivo de política de autorização

Esta é a forma como estabelecemos o arquivo de política de autorização para nosso exemplo:

```
grant
      Principal com.sun.security.auth.SolarisPrincipal "sdo"
      Principal com.sun.security.auth.SolarisNumericGroupPrincipal "45" (
      permission java.io.FilePermission "${/} files","read";
);
```

Esta entrada permitirá que um usuário leia em */files* apenas se o nome de *login* for "sdo" e se o usuário é um participante do grupo "45".

6.4. Executar o programa

Lembre-se de incluir os seguintes argumentos:

- -Djava.security.manager – para ativar a verificação de acessos
- -Djava.security.policy – para indicar o arquivo de políticas de segurança padrão
- -Djava.security.auth.policy – para indicar o arquivo de políticas de segurança do JAAS
- -Djava.security.auth.login.config – para indicar o arquivo de configuração do *Login*

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.