

Explorando serviços Java EE

Interceptadores e o Serviço de Tempo

Construiremos uma aplicação que realiza o agendamento e envio de mensagens em determinada data e hora utilizando recursos Java EE

BRUNO DANIEL MARINHO

A necessidade de agendamento de tarefas em sistemas corporativos é uma realidade, e um dos importantes recursos da especificação EJB. Antigamente não havia maneira de realizar agendamento de forma padronizada em nossas aplicações. Com o EJB 2.1 foi introduzido o primeiro sistema de agendamento ainda bem limitado. Na especificação do EJB 3.0 API foi expandida, porém ainda deixava a desejar por falta de formas mais sofisticadas de realizar os agendamentos. Atualmente o EJB 3.1 demonstra o amadurecimento da API, e nos oferece diversas formas elegantes para trabalhar com o agendamento de tarefas. Outra importante ferramenta incluída na especificação EJB 3.0 são os interceptadores e nos oferecem uma infinidade de possibilidades como será apresentado no artigo.

A aplicação e tecnologias

Para nossa aplicação utilizaremos o NetBeans IDE 6.9.1, que já vem integrado com o servidor de aplicação Java EE GlassFish e o framework JavaServer Faces 2.0, proporcionando assim uma fácil configuração do ambiente de trabalho. A aplicação realizará o agendamento do envio de mensagens e tem o propósito de demonstrar o uso do Serviço de Tempo (Timer Service) do container java EE, em nosso caso Glassfish. Implementaremos também uma auditoria simples demonstrando o uso dos Interceptadores(Interceptors). Trabalharemos com um Stateless Session Bean estudando seu ciclo de vida e os métodos de callback. Para a camada de apresentação utilizaremos o JavaServer Faces e realizaremos o controle de acesso a aplicação de forma simples com JAAS.

O que são Stateless Session Bean

Os Stateless Session Bean são utilizados para fornecer métodos de negócio que realizam tarefas específicas e não mantêm qualquer estado. Cada chamada de método é independente das chamadas anteriores, todos os dados necessários ao método deverão ser passados em seus parâmetros. Os Stateless Session Bean são mantidos em um pool no servidor de aplicação de forma que qualquer instância poderá atender qualquer solicitação da mesma forma. Vamos dar uma olhada em seu ciclo de vida (veja a **Figura 1**).

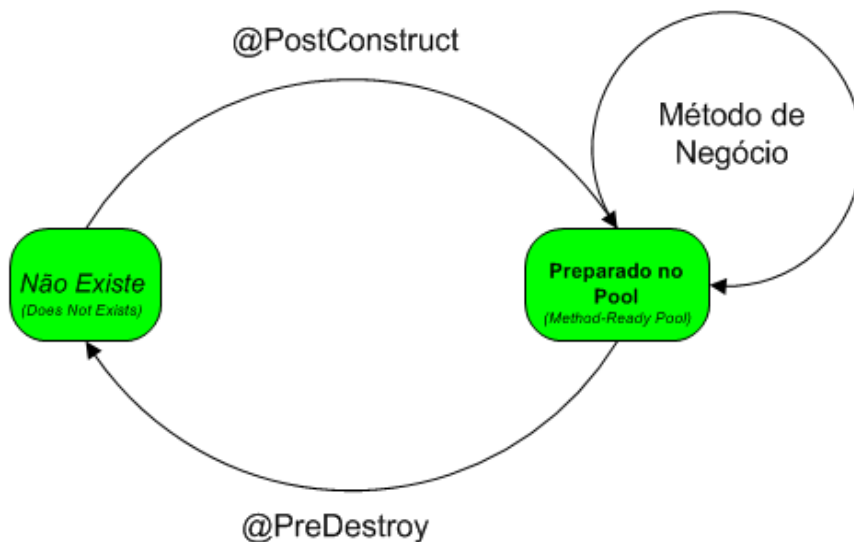


Figura 1. Ciclo de vida Stateless Session Bean.

Seus ciclo de vida é bem simples, consiste em apenas dois estados. O de inexistente onde ainda não foi instanciado pelo container e o de preparado no pool, onde esta já está pronto para atender as solicitações dos clientes. Na transição do estado de inexistente para o de preparado no pool podemos anotar um método com a anotação **@PostConstruct**, neste método de callback podemos inicializar algum recurso como uma conexão com o banco de dados. E no caminho de volta antes de destruir o bean podemos anotar um método com **@PreDestroy**, e realizar processamentos antes do container destruir o bean, como fechar a conexão com o banco de dados.

A aplicação Stateless Session Bean ao projeto

Nosso Stateless Session Bean será utilizado para prover as funcionalidades de agendamento e cancelamento de envio de mensagens. Inicialmente marcaremos um método chamado **inicio()** com **@PostConstruct** para que sejamos notificados no console cada vez que é criada uma nova instância deste Session Bean. Desejamos também manter uma estatística do número de realizados. Para esta tarefa utilizaremos uma variável de instancia em nosso EJB e a cada processamento incrementaremos esta variável. Para finalizar anotaremos o método **fim()** com **@PreDestroy** e antes do container destruir o Session Bean, ele nos mostrará o número de processamentos no console.

O que são Interceptadores

Um Interceptador (Interceptor) é uma classe POJO (*Plain Old Java Object*) que tem a capacidade de interceptar tanto chamadas de métodos de negócio quanto os eventos de ciclo de vida de um EJB. O ciclo de vida do interceptador é igual ao do tipo do EJB onde ele é aplicado, por exemplo, em nossa aplicação utilizamos um Stateless Session Bean então o interceptador tem exatamente os mesmos dois estados. Cada interceptador pode receber quaisquer recursos por injeção de dependência, em nosso caso utilizaremos a anotação **@Resource** para receber um objeto *SessionContext*, o qual nos fornece informações importantes para nossa auditoria.

Interceptadores podem manipular tanto os parâmetros de entrada dos métodos, como o retorno dos mesmos, isto pode ser útil para por exemplo validar esses parâmetros utilizando um serviço externo. Outra aplicação do uso de interceptadores é separar o que é de lógica de negócio de lógica funcional. Supondo que seja necessário que o retorno de um método de negócio seja compactado, este código de compactação não faz parte da regra negócio, então colocando em um interceptador podemos reutilizar o código de compactação e ao mesmo tempo manter nosso EJB coeso.

Outra importante funcionalidade dos interceptadores é de que podemos aplicá-los um conjunto de métodos particulares ou classes inteiras, oferecendo então um leve suporte a Orientação a Aspectos.

A aplicação dos Interceptadores ao projeto

Em nossa aplicação teremos duas tarefas para os interceptadores: um deles realizará a auditoria de nossos métodos de negocio. Será rastreado varias informações, como método invocado e a qual classe pertence, usuário que o chamou e a interface utilizada. Finalizando o interceptador irá sinalizar no console quando os métodos de callback forem executados no EJB.

Enquanto nosso outro interceptador terá a responsabilidade de marcar o tempo de execução dos métodos de negocio e notificar no console.

O serviço de tempo

A API Timer Service atualmente no EJB 3.1 evoluiu muito e nos disponibiliza hoje diversas maneiras de agendar tarefas. Atualmente podemos trabalhar com expressões através de **ScheduleExpression** que nos oferece uma sintaxe bem simples e flexível. Temporizadores automáticos são configurados com a anotação **@Schedule** e são agendados ao instanciar o bean.

Ao trabalhar com os métodos do objeto TimerService podemos agendar tarefas para serem executadas em determinada data e hora ou por intervalos de tempo. Os timers podem ser aplicados em uma serie de atividades comuns como a geração de relatórios e processamentos batch, realizar tarefas chave periodicamente como testes de conexão com serviços externos entre outros.

Obtendo as ferramentas

O Netbeans IDE 6.9.1 pode ser obtido entrando no site www.netbeans.org clicando no botão Download Free, você será redirecionado para uma tela com varias opções de download, escolha a opção java que venha com glassfish no momento de escrita do artigo o tamanho era 214 MB, lembrando que é necessário ter o JDK 6 para instalar o netbeans. Após o download execute o instalador, clique em personalizar e selecione as caixas de seleção Ide Base, Java SE, Java Web e EE, Glassfish clique em ok e depois em próximo ate finalizar a instalação.

Configurando o ambiente e criando o projeto

Para criar um novo projeto no NetBeans, acesse *Arquivo/Novo Projeto>Java EE>Aplicativo Corporativo*, digite o nome do projeto “AgendaMensagem”, clique em no botão próximo. Nesta tela você pode escolher o servidor de aplicação escolha o Glassfish Server 3, deixe marcado para o netbeans criar o modulo EJB e o modulo de aplicativo WEB. Neste momento o NetBeans criou três projetos que juntos compõe uma aplicação corporativa. (veja a **Figura 2**).

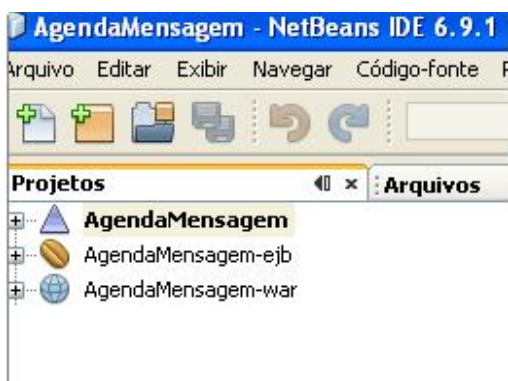


Figura 2. Três projetos no NetBeans.

O **projeto EJB** (AgendaMensagem-EJB) tem forma de feijão e empacota nossos Session Beans e interceptadores. O **projeto Web** (AgendaMensagem-WAR) com forma de globo empacota as páginas xhtml e os ManagedBeans do JSF.

Por fim o **projeto Enterprise (AgendaMensagem-EAR)** em forma de triângulo é um grande pacote. Seu conteúdo é formado pelo pacote WAR e o pacote EJB sendo ideal para distribuição da aplicação nos servidores de aplicação.

Criando o EJB

Para criar um novo Session Bean no netbeans clique com o botão direito no projeto EJB e escolha *Novo>Outro>Java EE>Bean de Sessão*. Vamos dar um nome ao Session Bean digite “**MeuSessionBean**” no nome do pacote digite “**br.com.agenda**” seleção do tipo de bean escolha a opção *sem estado (Stateless)* por fim marque para criar a interface local e finalize. O código inicial do nosso bean de sessão está apresentado na **Listagem 1**. Nesta primeira versão do nosso bean criamos um método de callback chamado **início()** e anotamos ele com **@PostConstruct** para que nós sejamos notificados no console cada vez que uma instância seja criada pelo container. Criamos nosso método de negocio chamado **processamento()** a principio ele imprimirá uma string simbólica, e a cada execução nós iremos incrementar nossa variável de instância controladora. Por fim criaremos outro método de callback método chamado **fim()** e iremos anotá-lo com **@PreDestroy** para que antes que o container destrua esta intancia ele nos notifique no console com a quantidade de processamentos que foram realizados. Quando se trabalha com EJBs o cliente sempre acessa o método de negocio através de uma interface. No nosso caso quanto criamos o EJB no netbeans marcamos para ele criar a interface local que é utilizada quando a aplicação é trabalha na JVM local. Então para finalizar necessitamos expor o método na interface local para isso crie uma interface chamada “**MeuSessionBeanLocal**” atualizando nosso código como mostrado na **Listagem 2**.

Listagem 1. Session Bean marcando os método de callback. MeuSessionBean.java

```
package br.com.agenda;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Stateless;

@Stateless
public class MeuSessionBean implements MeuSessionBeanLocal {

    private int processamentos;

    @PostConstruct
    private void inicio() {
        System.out.println("Criada uma nova instância [ " + this + " ]");
    }

    public void processamento() {
        System.out.println("Processando nosso ejb");
        processamentos++;
    }

    @PreDestroy
    private void fim() {
        System.out.print(" [ " + this + " ] realizou processamentos " + processamentos);
    }
}
```

Listagem 2. Interface Local com método exposto. MeuSessionBeanLocal.java

```
package br.com.agenda;
import javax.ejb.Local;

@Local
public interface MeuSessionBeanLocal {

    public void processamento();
}
```

Configurando o projeto web

Temos que configurar nosso projeto web para trabalhar com o JavaServer Faces, o netbeans vem integrado com os frameworks mais populares inclusive o JSF 2. Clique com o botão direito no projeto web **AgendaMessagem-war** escolha **Propriedades>Frameworks>Adicionar>JavaServer Faces>Ok**. Ao realizar esta configuração o JSF já vem configurado com **/faces/*** o que significa que temos que colocar **faces/** antes de acessar um arquivo, isto serve para que o servlet do JSF consiga trabalhar. No diretório **páginas Web** clique como botão direito em **Novo>Outro>Diretório** entre com o nome de “**site**” depois finalize, neste diretório ficarão as páginas da aplicação. O netbeans gera por padrão uma pagina **index.jsp no diretório páginas web neste momento pode exclui-la** pois a mesma não será utilizada. Agora criaremos uma pagina xhtml chamada **cadastro** que neste momento irá conter apenas um botão. O botão chama o método **cadastra()** no **MeuManagedBean** que criaremos a seguir, o método retorna apenas uma simples String com o conteúdo “**cadastro**” que é o nome da pagina sem a extensão xhtml, esse recurso é a navegação implícita presente no JSF 2. Clique com o botão direito no **diretório site>novos>outros>web>xhtml** digite o nome da pagina “**cadastro**”, ver código da página na **Listagem 3**.

Listagem 3. Primeira versão da tela de cadastro do sistema. Cadastro.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<title>Cadastro</title>
</h:head>
<h:body>
<h:form id="cadastro">
<h:commandButton value="Cadastrar" action="#{meuManagedBean.cadastra}" />
</h:form>
</h:body>
</html>
```

Criando a ManagedBean

O ManagedBean é a classe que fará a comunicação entre as paginas de nossa aplicação com o EJB. Marcamos o ManagedBean com escopo de sessão, utilizando a anotação **@EJB** e injetamos uma referencia a interface local de nosso Session Bean utilizando que nos disponibiliza o método **processamento()**. Criamos finalmente o método **cadastra()** no ManagedBean e este chama o metodo **processamento()** em nosso EJB.

Clique com botão direito no projeto web>**novos>outros>Java Server Faces>Bean gerenciado JSF** digite o nome do ManagedBean como **MeuManagedBean** digite também o nome do pacote **br.com.agenda.beans**, mude o escopo para session e **Finalize**. O código do MeuManagedBean é mostrado na Listagem 4.

Listagem 4. Managed bean invocando metodo no EJB. MeuManagedBean.java

```

package br.com.agenda.beans;

import br.com.agenda.MeuSessionBeanLocal;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.ejb.EJB;

@ManagedBean
@SessionScoped
public class MeuManagedBean {

    @EJB
    MeuSessionBeanLocal ejb;

    public MeuManagedBean() {
    }

    public String cadastra() {
        ejb.processamento();
        return "cadastro";
    }
}

```

*Iremos configurar a pagina inicial da aplicação para que seja sempre a pagina de cadastro. Altere o arquivo **web.xml** que está dentro do diretório **web-inf** e na linha de onde está `<welcome-file>faces/index.xhtml</welcome-file>` altere para `<welcome-file>faces/site/cadastro.xhtml</welcome-file>`.*

*. Chegou a hora de realizarmos o primeiro teste! Clique como botão direito no **AgendaMensagem-war** e escolha a opção executar, o netbeans irá iniciar o servidor e fazer o deploy da aplicação. Em alguns segundos o navegador abrirá com nossa tela de cadastro. Clique apenas duas vezes no botão cadastrar e retorne ao NetBeans na aba do GlassFish perto do console e pare o servidor clicando no x*

```

INFO: Criada uma nova instância [ br.com.agenda.MeuSessionBean@12155d2 ]
INFO: Processando nosso ejb
INFO: Processando nosso ejb
INFO: Iniciado servidor desligado
INFO: [ br.com.agenda.MeuSessionBean@12155d2 ] realizou processamentos 2

```

Figura 3. Resultado do teste do bean de sessão no console do glassfish

O resultado pode ser observado na **Figura 3**. Fica fácil de entender o funcionamento dos métodos de callback, verificando as notificações no console. Repare que da primeira vez que clicamos no botão cadastrar o servidor instanciou um bean e fomos notificados. Na segunda vez foi usado apenas um bean do pool e não criado um novo, somente o método de negócio foi executado. Para finalizar ao desligarmos o servidor antes de destruir nosso bean fomos notificados que ele realizou dois processamentos.

Adicionando Segurança a aplicação com JAAS

Posteriormente para que nosso interceptor possa ter acesso a identidade de quem invoca os métodos de negocio em nossos EJBs, e afim de gravar essa informação na auditoria. É necessário definir uma forma de acesso ao nosso sistema, definindo um grupo, um usuário e uma senha. Nós utilizaremos um *realm* de dados, que é basicamente a forma com que o container busca os dados do usuário para realizar a autenticação, no nosso caso optaremos pela forma simples de armazenamento em arquivo. No netbeans temos uma guia chamada **serviços**, esta guia prove acesso a recursos como por exemplo o banco de dados Derby Java DB e o GlassFish. Na guia **serviços>servidores>glassfish Server 3** clique com o botão direito e clique **visualizar console admin**. O navegador irá abrir e começara a carga do aplicativo, pode demorar um pouco é normal. O admin console do glassfish permite realizar toda configuração do container, nele podemos

visualizar vários itens no menu de opções, localize e expanda o item **segurança>domínios>novo domínio** defina o nome **agendaRealm**. Agora que criamos nosso realm temos que configura-lo , abaixo temos os tipos de realm de dados disponíveis em um combo, queremos o mais simples então escolha o item que termina em **FileRealm**.

No contexto JAAS temos que colocar o tipo de arquivo digitando **fileRealme** abaixo em **arquivo de tecla** digite `${com.sun.aas.instanceRoot}/config/agenda-keyfile` este é o nome e caminho do arquivo onde ficarão armazenados o usuário e a senha, clique no botão **Ok**. Voce ira voltar para a tela onde os realms são listados, localize **agendaRealm** e clique nele. Agora já podemos criar nosso usuário efetivamente, no canto superior esquerdo temos o botão **gerenciar usuários >novo** agora digite no campo id **agenda**, no grupo de usuário **Usuarios** e no campo de senha digite **123** depois clique em **Ok**, agora temos nosso container Java EE configurado.

Adicionando Segurança a aplicação

Neste momento, nossa aplicação tem que ser configurada para que possamos restringir o acesso a de usuários não autenticados as paginas do diretório **site**. Necessitamos de uma pagina de login que requisite nome de usuário e senha, e uma pagina de erro para quando o usuário falhar na autenticação ele ser informado. Para o login vamos utilizar a pagina `index.xhtml` que no caso já existe o netbeans havia criado ver código na **Listagem 7**, criaremos pagina de erro `erroLogin.xhtml` no diretório **paginas web** na ver **Listagem 8**. Acesse o arquivo `web.xml` dentro do diretório `web-inf` **adicione** as seguintes linhas da **Listagem 5** dentro da tag `<web-app>`. Na mesma pasta modifique o `sun-web.xml` realizando o mapeamento dos grupos de usuários com os papéis ver na **Listagem 6**. Nós definimos uma restrição de segurança para a pasta **site**, onde somente os usuários autenticados pelo container usando o **agendarealm** e que são **Papel-Usuarios** podem acessar os arquivos..

Agora nossa aplicação está protegida. Ao realizar o teste novamente será apresentado a pagina solicitado o login e senha para acessar as páginas dentro do diretório site.

Listagem 5. Linhas para adicionar no Web.xml

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Acesso</web-resource-name>
    <url-pattern>/faces/site/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description/>
    <role-name>Papel-Usuarios</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>agendaRealm</realm-name>
  <form-login-config>
    <form-login-page>/faces/index.xhtml</form-login-page>
    <form-error-page>/faces/erroLogin.xhtml</form-error-page>
  </form-login-config>
</login-config>
<security-role>
  <role-name>Papel-Usuarios</role-name>
</security-role>
```

Listagem 6. Código atualizado do Sun-Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD GlassFish Application Server 3.0 Servlet 3.0//EN" "http://www.sun.com/software/appserver/dtds/sun-web-app_3_0-0.dtd">
<sun-web-app error-url="">
  <context-root>/AgendaMensagem-war</context-root>
  <security-role-mapping>
    <role-name>Papel-Usuarios</role-name>
    <group-name>Usuarios</group-name>
  </security-role-mapping>
  <class-loader delegate="true" />
  <jsp-config>
```

```

    <property name="keepgenerated" value="true">
      <description>Keep a copy of the generated servlet class' java code.</description>
    </property>
  </jsp-config>
</sun-web-app>

```

Listagem 7. Formulário de login do sistema. index.xhtml

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>Login</title>
  </h:head>
  <h:body>
    <f:verbatim>
      Login no Sistema
      <hr/>
      <form method="post" action="#{request.contextPath}/j_security_check">
        Usuario <input type="text" id="username" name="j_username"/>
        Senha <input type="password" id="password" name="j_password"/>
        <input type="submit" value="Login"/>
        <br/>
        <hr/>
      </form>
    </f:verbatim>
  </h:body>
</html>

```

Listagem 8. Página de erro para o usuário. erroLogin.xhtml

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Login errado</title>
  </h:head>
  <h:body>
    <h:form>
      <h:outputText value="Usuario ou senha errada"/>
      <h:commandButton value="Realizar Login " action="index"/>
    </h:form>
  </h:body>
</html>

```

Codificando os interceptadores

Chegou a hora de começar a implementar nossa auditoria usando os **interceptadores**. A sintaxe de um interceptador consiste em uma classe que deve conter um método público anotado com **@AroundInvoke**, este mesmo método deve possuir um objeto **InvocationContext** como parâmetro, lançar a exceção **Exception** e retornar um **Object**, no primeiro interceptador de nossa aplicação iremos utilizar o **InvocationContext** para obter qual o método de negócio chamado e qual instância do EJB foi usada, o interceptador também pode obter recursos por injeção de dependência assim obteremos um **SessionContext** que nos fornece o usuário que esta chamando o método e a interface utilizada, quando o método `proceed()` do **invocationContext** é chamado o método interceptado será executado e nossa auditoria nos notifica antes e depois deste acontecimento, além disso nós colocamos uma notificação antes e depois de cada método do ciclo de vida ver código na **Listagem 9**, o segundo interceptador é mais simples e apenas marca o tempo da execução dos métodos ver **Listagem 10**, crie os dois interceptadores como classes java simples no mesmo pacote do **MeuSessionBean**.

Listagem 9. Primeira versão do Interceptador. MeuInterceptor.java

```

package br.com.agenda;
import javax.annotation.*;
import javax.ejb.SessionContext;

```

```

import javax.interceptor.*;
public class MeuInterceptador {
    @Resource
    private SessionContext context;
    @PostConstruct
    void inicio(InvocationContext ctx) throws Exception {
        System.out.println("Antes do @PostConstruct do EJB");
        ctx.proceed();
        System.out.println("Depois de @PostConstruct do EJB ");
    }
    @PreDestroy
    void fim(InvocationContext ctx) throws Exception {
        System.out.println("Antes do @PreDestroy o EJB");
        ctx.proceed();
        System.out.println("Depois do @PreDestroy o EJB ");
    }
    @AroundInvoke
    public Object auditoria(InvocationContext invocationContext) throws Exception {
        System.out.println("Interceptador executando antes do metodo...");
        System.out.println("Foi interceptado o Metodo " + invocationContext.getMethod() + " \n
chamado pelo Usuario [ " + context.getCallerPrincipal() + " ] " + " em uma instancia da classe " +
invocationContext.getTarget() + "\n utilizando a " + context.getInvokedBusinessInterface());
        Object proceed = invocationContext.proceed();
        System.out.print("Interceptador executando depois do metodo...");
        System.out.println("-----");
        return proceed;
    }
}

```

Listagem 10. Interceptador que marca o tempo. MeuTemporizador.java

```

package br.com.agenda;
import javax.interceptor.*;
public class MeuTemporizador {

    @AroundInvoke
    public Object marcaTempo(InvocationContext invocationContext) throws Exception {
        long comeco = System.currentTimeMillis();
        Object proceed = invocationContext.proceed();
        System.out.println("O Metodo levou " + (System.currentTimeMillis() - comeco) + "
Milissegundos para ser executado");
        return proceed;
    }
}

```

Aplicando os interceptadores

Os interceptadores podem ser aplicados á métodos individuais utilizando a anotação **@Interceptor(Class)** ou em todos os métodos de uma classe anotando a classe com **@Interceptors({Class[]})**, vamos alterar o **MeuSessionBean** e aplicar os interceptadores ver **Listagem 11**.

Listagem 11. Segunda versão do MeuSessionBean.java

```

Package br.com.agenda;
import javax.annotation.*;
import javax.ejb.Stateless;
import javax.interceptor.Interceptors;
@Stateless
@Interceptors({MeuInterceptador.class,MeuTemporizador.class})
public class MeuSessionBean implements MeuSessionBeanLocal {
    private int processamentos;
    @PostConstruct
    private void inicio() {
        System.out.print("Criada uma nova instancia [ " + this + " ]");
    }
    public void processamento() {
        System.out.println("Processando nosso ejb");
        processamentos++;
    }
}

```

```

    @PreDestroy
    private void fim() {
        System.out.print(" [ " + this + " ] realizou processamentos " + processamentos);
    }
}

```

Ao realizar esse novo teste, podemos verificar no console todas as informações sobre as chamadas de método (veja a Figura 4).

```

INFO: Antes do @PostConstruct do EJB
INFO: Criada uma nova instância [ br.com.agenda.MeuSessionBean@5c517a ]
INFO: Depois de @PostConstruct do EJB
INFO: Interceptador executando antes do metodo...
INFO: Foi interceptado o Metodo public void br.com.agenda.MeuSessionBean.processamento()
    chamado pelo Usuario [ agenda ] em uma instancia da classe br.com.agenda.MeuSessionBean@5c517a
    utilizando a interface br.com.agenda.MeuSessionBeanLocal
INFO: Processando nosso ejb
INFO: O Metodo levou 0 Milissegundos para ser executado
INFO: Interceptador executando depois do metodo...
INFO: -----

```

Figura 4. Chamada de método com os interceptadores aplicados.

Utilizando timer service

Chegou a hora em que finalmente vamos agendar nossos serviços, primeiramente vamos modificar `MeuSessionBean` novamente e injetar uma referencia ao `timerService` usando a anotação `@Resource`, modificaremos nosso método `processamento()` para que ele receba um objeto `Calendar` representando a data e hora e um objeto `Mensagem` que representa a mensagem que desejamos enviar, esse objeto deve ser criado no projeto EJB em um novo pacote `br.com.agenda.modelo` ver **Listagem 12**, dentro do método `processamento()` iremos utilizar o método `createTimer()` do objeto `timer` que é uma referencia injetada do `TimerService`, ele tem dois parâmetros um objeto `Date` contendo data e hora de execução e um objeto `Serializable` que usaremos para identificar o timer. Temos que criar um método para ser executado quando o timer atingir a data e hora desejada vamos anotá-lo com `@Timeout` no nosso caso é o método `tempo()` que recebe um timer como parâmetro, nós utilizamos o método `getInfo()` para recuperar a mensagem agendada, em nosso exemplo imprimimos no console suas informações, este código poderia ser trocado por um trecho real de envio de email. No `MeuSessionBean` criamos outro método chamado `listaTimer()` que retorna uma lista de mensagens e utiliza mais um método do objeto `timer` o `getTimers()` que nos retorna todos os timers agendados então extraímos as mensagens destes timers e devolvemos em forma de um lista de `Mensagens`, para finalizar temos o método `removeTimer()` que baseado em um id cancela um timer específico utilizando outro método do objeto `timer`, o `cancel()`. Vamos atualizar nosso código da interface local ver **Listagem 13**, e o código final do `MeuSessionBean` ver **Listagem 14**.

Listagem 12. Classe representando as mensagens. Mensagem.java

```

package br.com.agenda.modelo;
import java.io.Serializable;
import java.util.Date;

public class Mensagem implements Serializable {

    private int id;
    private String proprietario;
    private String mensagem;
    private boolean selecionado;
    private Date data;

    public Mensagem(int id, String proprietario, String mensagem, Date data) {
        this.id = id;
    }
}

```

```

        this.proprietario = proprietario;
        this.mensagem = mensagem;
        this.data = data;
    }

    public Date getData() {
        return data;
    }

    public void setData(Date data) {
        this.data = data;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getMensagem() {
        return mensagem;
    }

    public void setMensagem(String mensagem) {
        this.mensagem = mensagem;
    }

    public String getProprietario() {
        return proprietario;
    }

    public void setProprietario(String proprietario) {
        this.proprietario = proprietario;
    }

    public boolean isSelectedado() {
        return selecionado;
    }

    public void setSelectedado(boolean selecionado) {
        this.selecionado = selecionado;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Mensagem other = (Mensagem) obj;
        if (this.id != other.id) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 41 * hash + this.id;
        return hash;
    }
}

```

Listagem 13. Versao Final InterfaceLocal.java

```

package br.com.agenda;
import javax.ejb.Local;
@Local
public interface MeuSessionBeanLocal {

```

```

    public void processamento(br.com.agenda.modelo.Mensagem msg, java.util.Calendar c);

    public java.util.List<br.com.agenda.modelo.Mensagem> listaTimer();

    public void removeTimer(int id);

}

```

Listagem 14. Versão Final MeuSessionBean.java

```

package br.com.agenda;
import br.com.agenda.modelo.Mensagem;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.annotation.*;
import javax.ejb.*;
import javax.ejb.Timer;
import javax.interceptor.Interceptors;

@Stateless
@Interceptors({MeuInterceptador.class, MeuTemporizador.class})
public class MeuSessionBean implements MeuSessionBeanLocal {

    @Resource
    private TimerService timer;
    private int processamentos;

    @PostConstruct
    private void inicio() {


---


        System.out.print("Criada uma nova instancia [ " + this + " ]");
    }

    @Timeout
    public void tempo(Timer timerObj) {
        System.out.print("timer executado" + timerObj);

        Mensagem m = (Mensagem) timerObj.getInfo();
        SimpleDateFormat data = new SimpleDateFormat("dd/MM/yyyy HH:mm");
        System.out.print(" A Mensagem de id : " + m.getId() + " pertencente a o proprietario: " +
m.getProprietario()
            + " foi enviada em: " + data.format(m.getData()) + " com o conteudo: " +
m.getMensagem());
    }

    public void processamento(Mensagem msg, Calendar c) {
        timer.createTimer(c.getTime(), msg);
        processamentos++;
    }

    public List<Mensagem> listaTimer() {
        List<Mensagem> listaMensagens = new ArrayList<Mensagem>();


---



```

```

    Collection<Timer> lista = timer.getTimers();
    for (Timer t : lista) {
        Mensagem m = ((Mensagem) t.getInfo());
        if (m != null) {
            listaMensagens.add(m);
        }
    }
    return listaMensagens;
}

public void removeTimer(int id) {
    Collection<Timer> lista = timer.getTimers();
    for (Timer t : lista) {
        Mensagem m = ((Mensagem) t.getInfo());
        if (m.getId() == id) {
            t.cancel();
        }
    }
}

@PreDestroy
private void fim() {
    System.out.print(" [ " + this + " ] realizou processamentos " + processamentos);
}
}

```

Criando a pagina que gerencia os agendamentos

Em nosso projeto web AgendaMesagem-war temos que atualizar o código da página de *cadastro* para que ele permita o usuário digitar os campos da mensagem, assim como a data e hora desejada de envio ver na **Listagem 16**. Esta pagina chama o método `cadastro()` no `MeuManagedBean` que por sua vez chama o metodo de negocio `processamento()` passando como parâmetro um objeto **Calendar** representando a data e hora desejada e um objeto **Mensagem** construído com os dados que o usuário digitou no formulário ver **Listagem 15**. Dentro do diretório *site* crie o arquivo *grade.xhtml*, esta pagina será onde o usuário irá gerenciar os agendamentos, em uma tabela podemos visualizar os serviços agendados e cancelá-los convenientemente. Essa tela utiliza os métodos `listaTimer()` para preencher os dois componentes **datatable** que temos nesta pagina, um deles é onde visualizamos e podemos selecionar os timers para o cancelamento. O outro **datatable** exibe as mensagem selecionadas para exclusão quando clicado no botão excluir o método exclui do `MeuManagedBean` que por sua extrai o **id** das mensagens e chama o método de negocio `removeTimer()` cancelando os timer selecionados ver **Listagem 16**. Temos que atualizar nosso `MeuManagedBean` com os novos métodos utilizados nas paginas de cadastro e de grade ver **Listagem 17**.

Listagem 15. Versão Final cadastro.xhtml

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:c="http://java.sun.com/jsp/jstl/core">

```

```

<h:head>
  <title>Cadastro</title>
</h:head>
<h:body>
  <h:form id="cadastro">
    <h:outputText value="Agendamento de Eventos"/>
    <h:panelGrid>
      <h:panelGroup>
        <h:outputText value="id" />
        <h:inputText value="#{meuManagedBean.inputId}" />
      </h:panelGroup>
      <h:panelGroup>
        <h:outputText value="Nome" />
        <h:inputText value="#{meuManagedBean.inputNome}" />
      </h:panelGroup>
      <h:outputText value="Mensagem"/>
      <h:panelGroup>
        <h:inputTextarea value="#{meuManagedBean.inputMensagem}" />
      </h:panelGroup>
      <h:messages/>
    </h:panelGrid>
    <h:panelGrid columns="5">
      <h:panelGroup>
        <h:outputText value="Dia" />
        <h:inputText size="5" value="#{meuManagedBean.inputDia}" />
      </h:panelGroup>
      <h:panelGroup>
        <h:outputText value="Mes" />
        <h:inputText size="5" value="#{meuManagedBean.inputMes}" />
      </h:panelGroup>
      <h:panelGroup>
        <h:outputText value="Ano" />
        <h:inputText size="5" value="#{meuManagedBean.inputAno}" />
      </h:panelGroup>
      <h:panelGroup>
        <h:outputText value="Hora" />
        <h:inputText size="5" value="#{meuManagedBean.inputHora}" />
      </h:panelGroup>
      <h:panelGroup>
        <h:outputText value="Minuto" />
        <h:inputText size="5" value="#{meuManagedBean.inputMinuto}" />
      </h:panelGroup>
    </h:panelGrid>
    <h:commandButton value="Agendar" action="#{meuManagedBean.cadastrea}"/>
    <h:commandButton value="Grade" action="#{meuManagedBean.grade}"/>
  </h:form>
</h:body>
</html>

```

Listagem 16. Página de gerência de agendamentos .grade.xhtml

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
<h:head><title>Grade</title></h:head>
<h:body>
  <h:form id="grade">
    <h:outputText value="Grade de Eventos"/>
    <h:panelGrid>
      <h:dataTable id="items" border="3" value="#{meuManagedBean.listaTimer}" var="item">
        <h:column>
          <f:facet name="header">
            <h:outputText value="Selecione" />
          </f:facet>
          <h:selectBooleanCheckbox value="#{item.selecionado}" />
        </h:column>
        <h:column >
          <f:facet name="header">
            <h:outputText value="Id" />
          </f:facet>
          <h:outputText value="#{item.id}"/>
        </h:column>
        <h:column>

```

```

        <f:facet name="header">
            <h:outputText value="Proprietario"/>
        </f:facet>
        <h:outputText value="#{item.proprietario}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Mensagem"/>
        </f:facet>
        <h:outputText value="#{item.mensagem}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Data"/>
        </f:facet>
        <h:outputText value="#{item.data}">
        <f:convertDateTime timeZone="America/Sao_Paulo" pattern="dd/MM/yyyy HH:mm"/>
        </h:outputText>
    </h:column>
</h:dataTable>
</h:panelGrid>
<h:panelGrid>
<h:commandButton value="Selecionar para Exclusão" action="#{meuManagedBean.getItemSelecionados}"/>
</h:panelGrid>
<h:panelGrid>
    <h:outputText value="Mensagens para Apagar"/>
</h:panelGrid>
<h:dataTable id="nova" border="3" value="#{meuManagedBean.mensagemSelecionadas}" var="item">
    <h:column >
        <f:facet name="header">
            <h:outputText value="Id" />
        </f:facet>
        <h:outputText value="#{item.id}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Proprietario"/>
        </f:facet>
        <h:outputText value="#{item.proprietario}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Mensagem"/>
        </f:facet>
        <h:outputText value="#{item.mensagem}"/>
    </h:column>
</h:dataTable>
    <h:commandButton value="Cadastrar Novo" action="cadastro"/>
    <h:commandButton value="Excluir" action="#{meuManagedBean.exclui}"/>
</h:form>
</h:body>
</html>

```

Listagem 17. Versão Final MeuManagedBean.java

```

package br.com.agenda.beans;
import br.com.agenda.MeuSessionBeanLocal;
import br.com.agenda.modelo.Mensagem;
import java.io.Serializable;
import java.util.*;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean
@SessionScoped
public class MeuManagedBean implements Serializable {
    @EJB
    MeuSessionBeanLocal ejb;

    private Integer inputId, inputDia, inputMes, inputAno, inputHora, inputMinuto;
    private String inputNome, inputMensagem;
    private Mensagem selecionado;
    List<Mensagem> listaTimer;
    private List<Mensagem> mensagemSelecionadas;

    public String getItemSelecionados() {

```

```

        for (Mensagem msg : getListaTimer()) {
            if (msg.isSelecionado()) {
                if (!getMensagemSelecionadas().contains(msg)) {
                    getMensagemSelecionadas().add(msg);
                }
            } else {
                if (
                    (!(getMensagemSelecionadas() == null))
                    &&
                    (!(getMensagemSelecionadas().isEmpty()))
                ) {
                    getMensagemSelecionadas().remove(msg);
                }
            }
        }
        return "grade";
    }

    public String cadastra() {
        Calendar c = Calendar.getInstance();
        c.set(getInputAno(),getInputMes() - 1,getInputDia(),getInputHora(),getInputMinuto(), 00);
        Mensagem m = new Mensagem(getInputId(), getInputNome(), getInputMensagem(), c.getTime());
        ejb.processamento(m, c);
        setListaTimer(ejb.listaTimer());
        limparCampos();
        return "grade";
    }

    public String grade() {
        setListaTimer(ejb.listaTimer());
        limparCampos();
        return "grade";
    }

    public String exclui() {
        for (Mensagem m : getMensagemSelecionadas()) {
            ejb.removeTimer(m.getId());
        }
        setListaTimer(ejb.listaTimer());
        setMensagemSelecionadas(new ArrayList<Mensagem>());
        return "grade";
    }

    public void limparCampos(){
        setInputAno(null); setInputDia(null);
        setInputId(null); setInputMes(null);
        setInputHora(null);setInputMinuto(null);
        setInputNome(null); setInputMensagem(null);
    }

    public List<Mensagem> getMensagemSelecionadas() {
        if (mensagemSelecionadas == null) {
            mensagemSelecionadas = new ArrayList<Mensagem>();
        }
        return mensagemSelecionadas;
    }

    public void setMensagemSelecionadas(List<Mensagem> mensagemSelecionadas) {
        this.mensagemSelecionadas = mensagemSelecionadas;
    }

    public List<Mensagem> getListaTimer() {
        if (listaTimer == null) {
            listaTimer = ejb.listaTimer();
        }
        return listaTimer;
    }

    public void setListaTimer(List<Mensagem> listaTimer) {
        this.listaTimer = listaTimer;
    }

    public Integer getInputAno() {
        return inputAno;
    }

    public void setInputAno(Integer inputAno) {
        this.inputAno = inputAno;
    }

```

```

public Integer getInputDia() {
    return inputDia;
}

public void setInputDia(Integer inputDia) {
    this.inputDia = inputDia;
}

public Integer getInputHora() {
    return inputHora;
}

public void setInputHora(Integer inputHora) {
    this.inputHora = inputHora;
}

public Integer getInputId() {
    return inputId;
}

public void setInputId(Integer inputId) {
    this.inputId = inputId;
}

public String getInputMensagem() {
    return inputMensagem;
}

public void setInputMensagem(String inputMensagem) {
    this.inputMensagem = inputMensagem;
}

public Integer getInputMes() {
    return inputMes;
}

public void setInputMes(Integer inputMes) {
    this.inputMes = inputMes;
}

public Integer getInputMinuto() {
    return inputMinuto;
}

public void setInputMinuto(Integer inputMinuto) {
    this.inputMinuto = inputMinuto;
}

public String getInputNome() {
    return inputNome;
}

public void setInputNome(String inputNome) {
    this.inputNome = inputNome;
}

public Mensagem getSelecionado() {
    return selecionado;
}

public void setSelecionado(Mensagem selecionado) {
    this.selecionado = selecionado;
}
}

```

A **figura 5** demonstra a saída do sistema quando o timer é disparado, lembrando que este código poderia ser substituído pelo código real de envio de email por exemplo.

```

INFO: timer executadoTimer 3@1293201460631@@server@dcdomain1
INFO: A Mensagem de id :3 pertencente a o proprietario: Teste foi enviada em: 24/12/2010 12:40 com o conteudo: Mensagem para Teste

```

Figura 5. Notificação no console de um Timer disparado

Interface do sistema

Consiste na tela de *login* (veja a **Figura 6**) onde o usuário se autentica digitando usuário e senha . Essa tela é composta de um formulário tem características específicas para poder utilizar o JAAS são elas: O action do formulário deve conter como destino **j_security_check** o nome do campo de usuario deve ser **j_username** e o de senha **j_password**.

Login no Sistema

Usuario Senha

Figura 6. Tela de Login

Na tela de *cadastro* temos a entrada de dados da Mensagem data e hora.A tela consistem em simples componentes de input com suas variáveis no **MeuManagedBean** ao clicar no botão **grade** o método **grade()** retorna o outcome “**grade**” é simplesmente o nome da pagina desejada sem a extensão xhtml, lembrando que a navegação implícita só esta disponível a partir no JSF 2.(veja a **Figura 7**).

Agendamento de Eventos

id

Nome

Mensagem

Dia Mes Ano Hora Minuto

Figura 7. Tela de cadastro

A tela de *grade.xhtml* que permite ao usuário gerenciar os agendamentos apresenta duas tabelas e três botões .Uma das tabelas mostra os timers agendados, obtidos do EJB através do **MeuManagedBean** que permite a seleção de mensagens para exclusão. Ao clicar no botão “**selecionar para exclusão**” é chamado o método **getItemSelecionados()** do **MeuManagedBean** que analisa todas as mensagens da lista de mensagens que estão marcadas e adicionando-as em uma lista auxiliar que por fim são processadas pelo método **exclui()**. Responsável por extrair o id dessas mensagens selecionadas, cancelar as tarefas no EJB e atualizar a lista de timers com dados atualizados (veja a **Figura 8**),

Grade de Eventos

Selecione	Id	Proprietario	Mensagem	Data
<input type="checkbox"/>	1	javamagazine	Olá mundo	01/01/2012 13:00

Selecionar para Exclusão

Mensagens para Apagar

Id	Proprietario	Mensagem
<input type="text"/>	<input type="text"/>	<input type="text"/>

Cadastrar Novo Excluir

Figura 8. Tela de grade

Conclusões

Implementamos nossa aplicação explorando os Stateless Session Beans , para prover nossos métodos de negocio. Estudando seu ciclo de vida a aprendendo a usar seus métodos de callback. Conhecemos alguns recursos do Netbeans IDE que realmente não brinca em serviço, pois construímos uma aplicação JEE sem perder tempo correndo atrás de jars, mostrando sua forte integração com o Glassfish. Este por sua vez robusto e ao mesmo tempo elegante com seu console de administração apresentando as configurações de forma fácil e intuitiva. Configuramos a segurança utilizando um realm de dados e restringindo o acesso as paginas de nossa aplicação. Conhecemos os poderosos interceptadores e aplicamos com sucesso ao projeto realizando uma auditoria com uma boa quantidade de detalhes. Utilizando o Java Server Faces 2.0 nossas paginas tem a capacidade de interagir com o serviço de tempo agendando e cancelando tarefas. Estudamos como funciona e para que serve este importante recurso para as aplicações corporativas. Este artigo tem como objetivo mostrar na pratica o uso desses recursos, estes que são realmente muito importantes para que o desenvolvedor de sistemas corporativos aumente seu leque de ferramentas e busque aprimorar seus conhecimentos sobre JEE.