

Desenvolvendo um aplicativo completo com JSF, Facelets, Hibernate, Ajax e Design Patterns





Construir um aplicativo completo utilizando um conjunto extremamente atualizado de ferramentas e tecnologias.

- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**

- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**

- **Frameworks MVC** alavancaram na Web;
- **Struts, Hibernate, Tiles, Validator, XDoclet, Spring** são tecnologias extremamente adotadas no mercado atual;
- Tais tecnologias tornam o Java EE mais produtivo e fácil;
- Apesar de terem como base Java e Java EE, não fazem parte das especificações JCP;
- Desde 2005 os padrões vem sendo atualizados para “alcançar” estas tecnologias;

- **Resultado:**
 - **Struts + Validator = JSF**
 - **Hibernate = JPA**
 - **Tiles = Facelets**
 - **XDoclet = Annotations**
 - **JBoss Seam = Web Objects**
- **Conclusões: o que era bom, ficou ainda melhor e foi padronizado no JCP.**

- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**

O que é?

- Uma seleção confiável de ferramentas, frameworks e bibliotecas Java do mercado;
- Utilizada em produção na Globalcode;
- Suporte a multi-modelo de arquitetura, com e sem EJB, com e sem injeção de dependência;
- Esquema de segurança reforçado no JSF;
- Utiliza bibliotecas para user-interfaces ricas;

Composição

- **MVC e RAD: JSF**
- **Templating: Facelets**
- **Persistência: JPA com Hibernate;**
- **Rich interface: Scriptaculous;**
- **Bibliotecas de componentes JSF: Tomahawk, AJAX4JSF e RichFaces;**
- **Injeção de dependência: Jboss Seam;**

Composição

- **Segurança:**
 - Sem injeção / Seam: JSF-Security
 - Com Seam: esquema nativo do Seam;
- **Application Server: JBoss 4.0.5 e JBoss 4.2;**
- **Banco de Dados: MySQL**
- **Patterns Java EE: MVC, DAO, Factories, Business Delegate;**



- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**

- **Hibernate foi o framework padrão “de facto”;**
- **Conta com bons especialistas que simplificaram a visão de persistência comparado a dos Entity Beans;**
- **Seu único defeito era não ser do padrão Java EE;**
- **Expert Group do EJB 3.0 abandonou o EJB Entity Bean e criou a especificação Java Persistence API (JPA);**
- **JPA é muito semelhante ao Hibernate com xDoclet;**
- **JPA é fácil de usar;**
- **Hibernate rapidamente se “adaptou” para ficar compatível com JPA;**

- JPA trabalha com Annotations no lugar de xDoclet portanto requer Java SE 5;
- Utilização extremamente pequena de documentos XML;
- persistence.xml é o principal documento de configuração;
- Com JPA podemos trocar o framework de persistência sem afetar o aplicativo;
- O Hibernate disponibiliza configurações extras que podem violar esta afirmação;

- Exemplo de persistence.xml

```
<persistence-unit name="WebAppPU" transaction-type="RESOURCE_LOCAL">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <properties>
    <property name="hibernate.connection.url"
      value="jdbc:mysql://localhost/meubarco"/>
    <property name="hibernate.connection.driver_class"
      value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.connection.password" value="root"/>
    <property name="hibernate.connection.username" value="root"/>
    <property name="hibernate.dialect"
      value="org.hibernate.dialect.MySQLDialect"/>
    <property name="hibernate.cache.provider_class"
      value="org.hibernate.cache.NoCacheProvider"/>
  </properties>
</persistence-unit>
```

- Exemplo de JavaBean com JPA:

```
@Entity
@Table(name = "tipo_embarcacao")
public class TipoEmbarcacao implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name = "tipo_id", nullable = false)
    private Integer tipoId;
    @Column(name = "nome_tipo", nullable = false)
    private String nomeTipo;
    @Column(name = "imagem_grande", nullable = true)
    private String imagemGrande;
    @Column(name = "imagem_pequena", nullable = true)
    private String imagemPequena;
```


- Usando JPA:

```
public void exemploUsoNovo() {  
    TipoEmbarcacao objeto = new TipoEmbarcacao();  
    objeto.setNomeTipo("Exemplo uso JPA");  
    //Comunicação JPA  
    EntityManagerFactory emf;  
    emf = Persistence.createEntityManagerFactory("WebAppPU");  
    EntityManager em = emf.createEntityManager();  
    em.getTransaction().begin();  
    em.persist(objeto);  
    em.getTransaction().commit();  
    em.close(); //so para slide, deveria usar finally  
}
```

```
<persistence-unit name="WebAppPU" transaction-type="RESOURCE_LOCAL">  
    <provider>org.hibernate.ejb.HibernatePersistence</provider>  
    <properties>  
        <property name="hibernate.connection.url"  
            value="jdbc:mysql://localhost/meubarco"/>  
        <property name="hibernate.connection.driver_class"  
            value="com.mysql.jdbc.Driver"/>  
        <property name="hibernate.connection.password" value="root"/>  
        <property name="hibernate.connection.username" value="root"/>  
        <property name="hibernate.dialect"  
            value="org.hibernate.dialect.MySQLDialect"/>  
        <property name="hibernate.cache.provider_class"  
            value="org.hibernate.cache.NoCacheProvider"/>  
    </properties>  
</persistence-unit>
```

- Usando JPA:

```
public void exemploUsoLeitura() {  
    //Comunicação JPA  
    EntityManagerFactory emf;  
    emf = Persistence.createEntityManagerFactory("WebAppPU");  
    EntityManager em = emf.createEntityManager();  
    Query q = em.createQuery(  
        "select object(o) from Associado as o");  
    Collection<Associado> resultado =  
        (Collection <Associado>)q.getResultList();  
    em.close(); //so para slide, deveria usar finally  
}
```

- **Dicas de uso:**
 - **Cuide bem do equals e hashCode;**
 - **Utilize sempre que possível named queries;**
 - **Assim como JDBC, feche objetos no finally;**
 - **Quando utilizar JSF 1.2, Spring ou Seam, delegue a gestão de objetos EntityManager para o framework;**
 - **Configure bem seus relacionamentos e aproveite os recursos de collections, como lazy;**
 - **O Netbeans é uma excelente ferramenta para JPA;**

Criando classes JPA com engenharia reversa, utilizando NetBeans 5.5

- 🌀 Arquiteturas tradicionais Web;
- 🌀 GC Web 2.0 Toolkit;
- 🌀 Persistência: JPA com Hibernate
- 🌀 **MVC: JSF e Facelets**
- 🌀 User-interface: AJAX4JSF e Richfaces
- 🌀 Design-patterns: juntando tudo com estilo
- 🌀 Injeção de dependência: estava tudo errado?
- 🌀 Projeto mc31
- 🌀 Perguntas & Respostas

Sobre JavaServer Faces

- Do mesmo criador do Struts;
- Paradigma de programação visual de User-interfaces aplicado à web;
- É um framework que permite a criação de aplicações Web com semântica de Swing implementando MVC;
- “Toolability = Ferramentabilidade” ;
- É uma especificação Java EE – JSR 127(JSF 1.1)/JSR 252(JSF 1.2);
- Faces é mais fácil de aprender que Struts;
- Faces é mais componentizado;

J2EE Web Container

J2EE EJB Container

JavaServer Faces

Controller

Servlet

View

Java
Server
Pages

Custom
Tag

Model

JavaBeans

EJB

Página JSF

```
<f:view>
  <h:form>
    <h:outputText value="e-mail"/>
    <h:inputText value="#{olaMundoMB.email}" />
    <h:commandLink value="Chamar metodo no MB"
      action="#{olaMundoMB.metodo}" />
  </h:form>
</f:view>
```

faces-config.xml

```
<managed-bean>
  <managed-bean-name>
    olaMundoMB
  </managed-bean-name>
  <managed-bean-class>
    ...OlaMundoMB
  </managed-bean-class>
  <managed-bean-scope>
    request
  </managed-bean-scope>
</managed-bean>
```

Managed Bean

```
public class OlaMundoMB {
  private String email;
  public void metodo() {
    System.out.println("Chamada ao MB...");
    System.out.println("Email lido:" + email);
  }
  public String getEmail() {
    return email;
  }
  public void setEmail(String email) {
    this.email = email;
  }
}
```


OlaMundo JSF com NetBeans 5.5

Componentes para JavaServer Faces

- **DataGrid;**
- **Tabbed Panel;**
- **PanelGrid;**
- **SelectOneMenu, SelectOneRadio, SelectOneListBox;**
- **SelectManyMenu, SelectManyRadio, SelectManyListBox;**
- **FileUpload;**
- **Auto-complete AJAX;**
- **Muitos outros...**

JavaServer Faces Vs. Struts

- **Struts trabalha com classes “Action” e classes “Form”;**
- **JSF utiliza o MB para agrupar as Actions e dados do form;**
- **Leitura e validação de dados é muito mais fácil com JSF;**
- **XML de configuração do JSF é mais enxuto;**
- **JSF trabalha com conceito de RenderKit separado dos componentes. Na prática um mesmo JSP pode gerar diferentes saídas: HTML, WML, XML, etc;**
- **Struts é “preso” em HTML.**

Mais sobre JavaServer Faces

- Existe um mini-curso específico com material para download;
- A Globalcode é pioneira no uso e no ensino de JSF;
- Fazemos parte do expert group da especificação JCP;

Sobre Facelets

- Esquema de templating semelhante ao Tiles do Struts;
- Facilita reaproveitamento de estrutura de telas;
- Facilita o reuso de pedaços de telas;
- Conceito de “herança de telas”;
- Será parte do JSF 2.0;
- Não tem porque não usar...
- DEMO: `modelo1.xhtml` – `associado.xhtml`

- 🌀 Arquiteturas tradicionais Web;
- 🌀 GC Web 2.0 Toolkit;
- 🌀 Persistência: JPA com Hibernate
- 🌀 MVC: JSF e Facelets
- 🌀 User-interface: **AJAX4JSF** e **Richfaces**
- 🌀 Design-patterns: juntando tudo com estilo
- 🌀 Injeção de dependência: estava tudo errado?
- 🌀 Projeto mc31
- 🌀 Perguntas & Respostas

- **Inúmeros componentes JSF para AJAX disponíveis no mercado;**
- **AJAX4JSF e RichFaces ganham pela facilidade de uso:**
 - Adicionar os JARs no WEB-INF/lib;
 - Configurar um servlet-filter no web.xml;
 - Usar componentes via custom tags nas páginas;
- **DEMO: apresentação de Tabbed Panel AJAX e Suggestion-box.**

- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**

- **JSF** tende a centralizar código no managed bean;
- **Patterns** que devemos utilizar com **JSF**:
 - **Data Access Object**: centralizar acesso ao JPA
 - **Business Delegate**: chama EJBs e WS de negócio;
 - **Factory**: caso tenha arquiteturas múltiplas, com e sem EJBs, com e sem JPA, etc.
 - **Adapter e Bridge**: indicados para integrações com legados Java;

- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**

- Representa uma forma de inversão de controle;
- Trabalha com idéia de receber objetos e não buscar objeto;
- Exemplo: para utilizar JPA buscamos objeto

EntityManager:

```
public void exemploUsoNovo() {  
    TipoEmbarcacao objeto = new TipoEmbarcacao();  
    objeto.setNomeTipo("Exemplo uso JPA");  
    //Comunicação JPA  
    EntityManagerFactory emf;  
    emf = Persistence.createEntityManagerFactory("WebAppPU");  
    EntityManager em = emf.createEntityManager();  
    em.getTransaction().begin();  
    em.persist(objeto);  
    em.getTransaction().commit();  
    em.close(); //so para slide, deveria usar finally  
}
```

- O framework, através de annotations, encontrará qual objeto deve injetar:

```
@Stateful
public class TurmasManagerBean {

    @PersistenceContext()
    EntityManager em;

    public void exemploUsoNovo() {
        TipoEmbarcacao objeto = new TipoEmbarcacao();
        objeto.setNomeTipo("Exemplo uso JPA")
        em.persist(objeto);
    }
}
```

- **Alguma injeção no JSF é suportada no Java EE 5;**
- **Para ganhar mais recursos de injeção, devemos usar um framework adicional;**
- **Atualmente os melhores são Jboss Seam e Spring;**
- **JBoss Seam vai virar JSR, WebObjects;.**

- 🌀 Arquiteturas tradicionais Web;
- 🌀 GC Web 2.0 Toolkit;
- 🌀 Persistência: JPA com Hibernate
- 🌀 MVC: JSF e Facelets
- 🌀 User-interface: AJAX4JSF e Richfaces
- 🌀 Design-patterns: juntando tudo com estilo
- 🌀 Injeção de dependência: estava tudo errado?
- 🌀 **Projeto mc31**
- 🌀 Perguntas & Respostas



- Requisitos para execução:
 - NetBeans 5.5.x ou superior;
 - Jboss 4.2.2 ou superior;
 - MySQL, podendo ser alterado para outros dbs
 - Hibernate 3.2 Entity Manager
 - Plug-in de suporte a Facelets para NetBeans
- Endereços de download:
 - www.netbeans.org
 - www.jboss.org
 - www.mysql.com
 - www.hibernate.org
 - <https://nbfaceletssupport.dev.java.net/features.html>

- **Passo 1: instalações**
 - **Instalar NetBeans 5.5;**
 - **Instalar JBoss com suporte a EJB3;**
 - **Instalar MySQL**
 - **Descompactar o Hibernate Entity Manager;**
- **Passo 2: criar Database e Tabela no MySQL**

```
CREATE DATABASE mc31;  
CREATE TABLE associado (  
    codigo_associado int(10) unsigned NOT NULL auto_increment,  
    nome_associado varchar(255) NOT NULL,  
    senha varchar(45) NOT NULL, `email` varchar(45) NOT NULL,  
    ativo tinyint(3) unsigned NOT NULL, `data_cadastro` datetime NOT NULL,  
    codigo_ativacao varchar(10) default NULL,  
    cidade varchar(60) default NULL, `estado` varchar(2) default NULL,  
    PRIMARY KEY (`codigo_associado`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```


- **Passo 3 – Integrar JBoss com NetBeans**

- Clique em Tools -> Server Manager, em seguida em Add Server... Escolha como servidor Jboss Application Server e em seguida aponte o diretório de instalação.

- **Passo 4 – Criar biblioteca Hibernate no Netbeans**

- Criar banco de dados no MySQL e tabela conforme comando apresentado;
- Dentro do NetBeans vá no menu Tools -> Library Manager em seguida clique em New Library, com nome Hibernate.
- Clique em Add Jar/Folder... e escolha os seguintes JARs no diretório onde está instalado o Hibernate Entity Manager.
- Escolha os JARS hibernate-entitymanager.jar, lib\ejb3-persistence.jar e lib\hibernate-annotations.jar.

- **Passo 5 – Instalar plug-in Facelets no NetBeans**

- Clique em Tools -> Update Center e em seguida escolha Install Manually. Escolha todos os arquivos com extensão .nbm no local onde você descompactou o plug-in nbfaceletssupport-0-5.zip.
- Prosiga com a instalação respondendo ao Agreement.

- **Passo 6 – Projeto NetBeans**

- Abra o projeto NetBeans mc31;
- Em Configuration Files, abra o arquivo persistence.xml e altere as credenciais do seu banco de dados e nome de database.
- Clique em Run para o NetBeans fazer o build e deploy do projeto.

- **Final:**

- Pronto! Agora navegue no aplicativo pela URL
<http://localhost:8080/mc31/associados.jsf>

- 🌀 **Arquiteturas tradicionais Web;**
- 🌀 **GC Web 2.0 Toolkit;**
- 🌀 **Persistência: JPA com Hibernate**
- 🌀 **MVC: JSF e Facelets**
- 🌀 **User-interface: AJAX4JSF e Richfaces**
- 🌀 **Design-patterns: juntando tudo com estilo**
- 🌀 **Injeção de dependência: estava tudo errado?**
- 🌀 **Projeto mc31**
- 🌀 **Perguntas & Respostas**



Agradecemos a presença!



Iniciativa Open4Education