

11

J820

Geração de código com xdoclet

Sobre este módulo

- *Este módulo apresentará o **XDoclet** - uma ferramenta usada como tarefa do Ant que permite criar e executar templates para gerar qualquer tipo de texto, inclusive XML e Java*
- *Mostraremos dois exemplos*
 - ***Uso genérico de XDoclet:** como usar XDoclet para construir e preencher qualquer template*
 - ***Uso prático de XDoclet:** exemplo de montagem de aplicação EJB usando Entity Beans (BMP) e Session Beans com geração de vários artefatos*

O que é XDoclet

- É uma ferramenta para gerar texto e código através do processamento de templates.
 - É uma extensão da **API Javadoc Doclet**, da Sun
- Aplicação mais popular: gerar artefatos de EJB
 - Criar um EJB é uma tarefa trabalhosa por envolver vários arquivos de configuração em XML e diversas classes
- Gera texto através do Ant e a partir do código-fonte
 - Requer uma **fonte** e um **template** para funcionar
 - Configuração do texto a ser gerado pode ser feito no arquivo-fonte, através de **tags nos comentários** ou passando parâmetros através do Ant
 - É possível usar XDoclet e nunca criar um template usando as tarefas pré-configuradas (EJB, Web, etc.)

Como usar XDoclet

- Para configurar o ambiente de desenvolvimento e projeto para usar XDoclet é preciso
 - Colocar o arquivo **xdoclet.jar** e dependências em lugar acessível (no diretório **lib/** do projeto, por exemplo)
 - Decidir sobre o nome de um diretório onde colocar as fontes geradas (usamos **gensrc/**)
 - Alterar o alvo "**compile**" de build.xml para que procure arquivos de código-fonte também em **gensrc/**
 - Criar alvo para **apagar** arquivos gerados
- Depois, é preciso configurar o Ant para reconhecê-lo
 - Definir uma **tarefa nova** com base em uma DocletTask do pacote xdoclet.jar: **EJBDocletTask**, **WebDocletTask**, **DocumentDocletTask**, etc.

Como usar XDoclet através do Ant

- Primeiro escolha uma das suas tarefas para o Ant
- As quatro principais tarefas são
 - `xdoclet.DocletTask` - tarefa genérica; serve de superclasse para as outras
 - `xdoclet.doc.DocumentDocletTask`
 - `xdoclet.ejb.EjbDocletTask`
 - `xdoclet.web.WebDocletTask`
- Cada tag tem um conjunto de atributos aceitos e um conjunto de tags internos com seus atributos
 - Consulte o manual de referência para detalhes. Veja no diretório `doc/` da distribuição.
- Depois crie um novo tag usando a tarefa `<taskdef>`

Definição de uma tarefa usando `<taskdef>`

- O exemplo abaixo usa o **DocletTask**, a tarefa mais simples. Use-o para processar seus próprios templates

```
<taskdef name="xdoclet"
         classname="xdoclet.DocletTask">
  <classpath refid="xdoclet.path" />
</taskdef>
```

↖ *xdoclet.jar e log4j.jar*

- Na distribuição 1.2 beta 3, o **xdoclet.path** acima deve conter, além do arquivo **xdoclet.jar**
 - O arquivo **log4j.jar** (ou equivalente, da aplicação Log 4J) em qualquer aplicação do XDoclet
 - O arquivo **j2ee.jar** ou equivalente, em tarefas EJBDoclet
 - O arquivo **servlet.jar**/equivalente, em tarefas WebDoclet

Uso de `<xdoclet>`

- *DocletTask* requer um sub-elemento `<template>` que descreve a geração de texto segundo determinado template.
 - *Você pode escrever um template do zero, utilizar outro como base ou encontrar um template pronto que faça o que você quer.*
 - *Há vários templates prontos no JAR do XDoclet*
- `<template>` deve informar a localização do arquivo de template e o nome do arquivo (ou arquivos) resultante(s)
 - *Se nome do arquivo de resultados tiver um `{0}`, o processamento será repetido para cada arquivo encontrado e `{0}` será substituído pelo nome do arquivo original sem a extensão.*

```
<xdoclet sourcepath="${src.dir};${gen.src.dir}" destdir="tmp">
  <classpath refid="xdoclet.path" />
  <fileset dir="${src.dir}">
    <include name="**/*.java" />
  </fileset>
  <template templateFile="lib/business.delegate.j"
    destinationFile="result-{0}.txt" />
</xdoclet>
```

Além do `xdoclet.path`, deve haver um path para dependências dos arquivos processados

Execução de XDoclet

- XDoclet é executado através da execução de um alvo do Ant que contenha a tarefa criada. Para cada template, ele irá
 - *Carregar as classes* envolvidas e passá-las para o Javadoc que irá extrair informações de sua estrutura
 - *Ler os tags* incluídos nos comentários
 - *Ler parâmetros* passados pelo Ant (e possivelmente sobrepor valores de alguns tags)
 - *Preencher o template*
- Código-fonte deve ser gerado para que possa ser imediatamente compilado
 - Alterações devem ser feitas nos templates, Ant, comentários, etc. e *não no código gerado*

Templates

- XDoclet oferece uma vasta **biblioteca de tags** usados para criar templates
- Arquivos de template convencionalmente possuem extensão "***.j**" e se assemelham, na estrutura, a documentos JSP
 - Os tags de templates obedecem a XML
 - Tags de templates permitem inserir variáveis de configuração, todas as informações de classes, métodos, tipos, dados de arquivos, tags do Javadoc, etc.
- Para as suas principais tarefas (EJB Doclet e Web Doclet), XDoclet oferece vários **templates prontos**
 - Localize-os em **xdoclet.jar** e veja seu conteúdo

Estrutura dos templates

- Todo template tem um **namespace** iniciado por XDt
 - **XDtClass**
 - **XDtMethod**
 - **XDtConfig**
 - **XDtStrutsForm**
 - ...
- Cada namespace possui um vocabulário de **nomes de tags** e **atributos**
 - Tags têm o formato **<XDtNamespace:nome ... >**.
Exemplo: **<XDtClass:className />**
 - Alguns tags imprimem **conteúdo**. Geralmente são tags **vazios** (mas podem conter atributos)
 - Outros tags **podem conter** tags e são usados em repetições, condicionais, etc.

Há 40 namespaces e 222 tags definidos na distribuição XDoclet 1.2 beta 3

Exemplo de um template

- O template abaixo requer um FileSet contendo classes. Ele irá gerar um arquivo com os métodos de cada classe encontrada

```
Classe: <XDtClass:fullClassName />
Pacote: <XDtPackage:packageOf><XDtClass:fullClassName />
        </XDtPackage:packageOf>
<XDtMethod:forAllMethods>
    Metodo: <XDtMethod:methodName />
            <XDtMethod:methodName superclasses="false" />
            (<XDtParameter:parameterList
                includeDefinition="true" />)
</XDtMethod:forAllMethods>
```

- Eis um possível resultado, em um arquivo {0}.txt

```
Classe: teste.exemplo.MeuObjeto
Pacote: teste.exemplo
        Metodo: void
                imprimir
                ()
        Metodo: boolean
                gravar
                (String texto, boolean backup)
```

Para usar
XDoclet não é
necessário
escrever
templates!

//

Parâmetros e tags

- Se o template os utiliza, pode-se passar **parâmetros de configuração** através do **Ant**, e outros parâmetros através dos **tags nos Javadocs**. O seguinte template

```
Param: <XDtConfig:configParameterValue  
                                paramName="date" />
```

```
Tag: <XDtClass:classTagValue  
      tagName="teste:um" paramName="nome"/>
```

imprime valor do parâmetro **date** passado no build.xml

```
<template templateFile="lib/test.j"  
          destinationFile="result-{0}.txt">  
  <configParam name="date" value="${DSTAMP}" />  
</template>
```

e o atributo **nome** de um tag definido nos JavaDocs

```
/** @teste:um nome="Isto é um teste" */
```

EJBDocletTask

- A mais popular tarefa de XDoclet é **EJBDocletTask**
 - Não é preciso aprender a fazer templates para usá-la
 - Define uma coleção de tags utilizáveis nos JavaDocs
- Com EJBDoclet, é possível escrever apenas uma classe: o **Enterprise Bean**, e gerar automaticamente
 - As duas interfaces remotas: Home e componente
 - As duas interfaces locais: LocalHome e Local
 - O deployment descriptor ejb-jar.xml
 - XMLs de fabricantes como JBoss (jboss.xml) e outros
 - DAOs, Value Objects, Service Locators, etc.
- É possível ainda ir além e escrever seu próprio template, se algum não construir os artefatos como esperado, ou para gerar artefatos adicionais.

Uso de EJBDocletTask no Ant

- Primeiro, defina a tarefa

```
<taskdef name="ejbdoclet"  
         classname="xdoclet.ejb.EjbDocletTask">  
  <classpath refid="xdoclet.path" />  
</taskdef>
```

- Depois, use-a

```
<ejbdoclet sourcepath="src" destdir="${src.gen.dir}"  
           classpathref="xdoclet.path" ejbspec="2.0">  
  <fileset dir="src">  
    <include name="**/*Bean.java" />  
  </fileset>  
  <remoteinterface/>  
  <homeinterface/>  
  <utilobject/>  
  <entitypk/>  
  <dataobject/>  
  <deploymentdescriptor destdir="${meta.dir}"/>  
  <jboss datasource="java:/OracleDS" />  
</ejbdoclet>
```

Arquivos que serão processados pelos templates

Cada sub-elemento (todos são opcionais) é um comando <template> com template pré-definido

Algumas subtarefas de EJBDocletTask

- As subtarefas são extensões do elemento `<template>` com um arquivo de template previamente definido
- Esta lista está resumida. Consulte a documentação para informações sobre namespaces e atributos
 - `<dataobject/>`**: cria Data Transfer Object (Value Object)
 - `<deploymentdescriptor/>`**: cria ejb-jar.xml padrão
 - `<entitypk/>`**: cria chave primária (somente entity beans)
 - `<homeinterface/>`**: cria interface home RMI-IIOP
 - `<localhomeinterface/>`**: cria interface home local
 - `<localinterface/>`**: cria interface do objeto local
 - `<remoteinterface/>`**: cria interface do objeto remoto
- Tag que gera deployment descriptor do JBoss
 - `<jboss>`**: cria jboss.xml, jbossjdbc-cmp.xml e jaws.xml

Configuração usando tags do Javadoc

- *Informações necessárias para o gerar deployment descriptor padrão e arquivos proprietários do servidor podem ser inseridas nos comentários Javadoc (antes de classes e métodos)*

```
/**
 * @ejb:bean    name="Produto"
 *              jndi-name="ejb/loja/produto"
 *              type="BMP"
 *              view-type="both"
 * @ejb:dao     class="loja.integration.ProdutoDAO"
 *              impl-class="loja.integration.JdbcProdutoDAO"
 * @ejb:transaction type="Required" ← default, para
 * @ejb:resource-ref todos os métodos
 *              res-name="jdbc/LojaDB"
 *              res-type="javax.sql.DataSource"
 *              res-auth="Container"
 * @jboss:resource-manager
 *              res-man-class="javax.sql.DataSource"
 *              res-man-name="jdbc/LojaDB"
 *              res-man-jndi-name="java:/DefaultDS"
 */
public class ProdutoBean implements EntityBean { ... }
```

Alguns Javadoc tags do EJBDoclet

- Use nos comentários Javadoc antes da declaração de classe e/ou antes dos métodos. Consulte a documentação sobre atributos.
 - **@ejb:bean** Possui vários atributos com informações sobre o bean que devem ser inseridas no seu deployment descriptor
 - **@ejb:create-method** Usado antes cada método create()
 - **@ejb:data-object** Permite configurar um Data-transfer object (value object) object para o entity bean
 - **@ejb:ejb-ref** Constrói elemento <ejb-ref>
 - **@ejb:ejb-transaction** Usado antes de cada método para definir atributos transacionais. Também define default para a classe
 - **@ejb:env-entry** Constrói elemento <env-entry>
 - **@ejb:interface-method** Usado antes de métodos de interface
 - **@ejb:permission** Define que papéis têm acesso ao método
 - **@ejb:persistent-field** Declara métodos que são campos CMP
 - **@ejb:resource-ref** Declara referências de recursos (BDs, filas)
 - **@ejb:select** Indica que método é um ejbSelect()

Como gerar artefatos com EJBDoclet

- Para gerar **todos** os artefatos de um EJB, só é necessário dispor da classe do bean
 - Primeiro, configure as informações do deployment descriptor (DD) no Javadoc da classe definindo os **@tags**
 - Cada **@tag** pode ter atributos na forma **nome="valor"**
- O tag **@ejb:bean** é o único obrigatório
 - Se ele for omitido, a geração funcionará, mas usará defaults que podem gerar dados inconsistentes

/**

Pode ser 'Stateful', 'Stateless'
ou 'BMP', 'CMP' de acordo
com o tipo de bean

```
name="AdminLojaSessionFacade"  
jndi-name="ejb/loja/admin"  
view-type="remote"  
type="Stateless"
```

Pode ser 'remote',
'local' ou 'both'

*/

```
public class AdminLojaBean implements SessionBean {}
```

EJBDoclet: @tags em métodos

- Alguns atributos podem ser configurados em métodos individuais. Estes são definidos nos comentários javadoc de cada método
 - @ejb:interface-method** [view-type="remote|local|both"]
 - Se método deve ser ou não exposto na interface do componente (remota, local ou ambas). Default: both
 - @ejb:create-method**
 - Se método é um método ejbCreate
 - @ejb:finder-method**
 - Se método é um ejbFindBy
 - @ejb:transaction** type="RequiresNew"
 - Atributo de política transacional para este método. Default é o valor definido para toda a classe.
 - @ejb:permission** role="nome"
 - Define um papel que pode ter acesso a este método. Default é o valor definido para toda a classe (se nenhum for definido, os tags <method-permission> não serão gerados)

EJBDoclet: outros artefatos

- Existem tags para gerar arquivos de configuração para os vários servidores de aplicação do mercado
 - JBoss: subtask <jboss>, javadoc namespace @jboss
 - Macromedia JRun: subtask <jrun>, javadoc namespace @jrun
 - BEA Weblogic: subtask <weblogic>, namespace @weblogic
 - IBM WebSphere: subtask <websphere>, namespace @websphere
- Como EJBDoclet estende Doclet, também suporta <template>, que pode ser usado para rodar templates adicionais

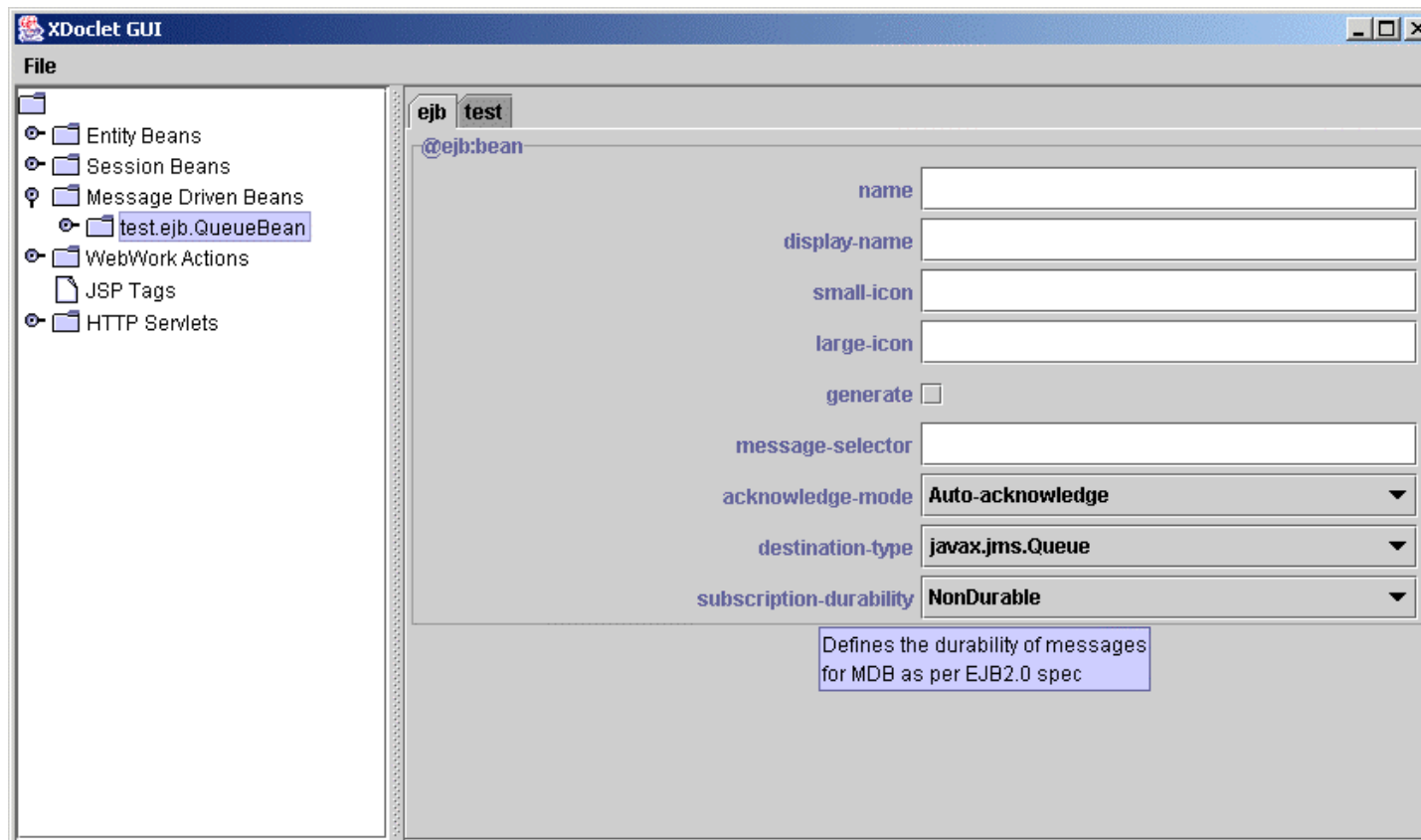
```
<ejbdoclet sourcepath="${src.dir};${gen.src.dir}"
           destdir="${tmp.dir}/genbd" ejbspec="2.0">
  <classpath refid="xdoclet.path" />
  <fileset dir="${src.dir}">
    <include name="**/*Bean.java" />
  </fileset>
  <template templateFile="${lib.dir}/business.delegate.j"
            destinationFile="{0}BusinessDelegate.java"
            ofType="javax.ejb.SessionBean"/>
</ejbdoclet>
```

WebDocletTask

- A tarefa **WebDocletTask** ajuda a construir dois deployment descriptors padrão para aplicações Web
 - *web.xml*
 - *arquivo.tld* (Tag Library Descriptor)
- Há suporte para o JRun e Struts
 - *Gera automaticamente o struts-config.xml*
- Há dois namespaces para os @tags do Javadoc
 - **@jsp**
 - **@web**
- Os comentários Javadoc devem ser incluídos nos arquivos Action e Form do Struts ou em servlets

Ferramentas

- No mesmo site do XDoclet, pode ser baixado o projeto *xdocletgui* (via CVS) que contém uma aplicação (ou plug-in para IDEs) que ajuda a construir os @tags



- *Veja as seguintes aplicações exemplo*
- ***XDoclet Demo***
 - *Mostra um exemplo simples gerando um documento a partir de um template que imprime os nomes de classes e monta seus métodos*
 - Use **ant test.xdoclet**
- ***EJBDoclet Demo***
 - *Mostra um exemplo de uma aplicação EJB simples consistindo de dois beans (Session e Entity) usando BMP e patterns (Business Delegate, DAO, Service Locator e Value Object)*
 - Use **ant xdoclet-code-gen**

- 1. *Adapte o build.xml da aplicação EJB fornecida para que utilize XDoclet*
 - *A aplicação contém apenas os beans e a aplicação cliente, que é uma página JSP*
 - *Utilize as informações nos comentários (EJB-QL, nomes JNDI, etc. para definir os tags)*
 - *Consulte a documentação do EJBDoclet*
 - a) *Inclua um <taskdef> para definir a tarefa <ejbdoclet>*
 - b) *Preencha o alvo gerar-fontes-xdoclet com uma ou mais tarefas <ejbdoclet> necessárias para gerar as interfaces remote para o session bean, local para o entity bean, primary key, ejb-jar.xml, jboss.xml, etc.*
 - c) *Teste a aplicação. Use ant deploy*

- [1] *Documentação XDoclet*. xdoclet.sourceforge.net
- [2] Erik Hatcher. *Java Development with Ant*. Manning, 2003
- [3] Eric Burke & Brian Coyner. *Java eXtreme Programming Cookbook*. O'Reilly, 2003

Curso J820
Produtividade e Qualidade em Java:
Ferramentas e Metodologias

Revisão 1.1

© 2002, 2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br