SGB_MANUAL_ConfiguracaoHibernate

Versão 1.0

Histórico de Revisões

Nome	Alterações	Data	Versão
Bruno Marquete	Criação do Documento	31/10/2012	0.1

Bruno Marquete	Criação da Introdução	31/10/2012	0.2
Bruno Marquete	Instalação do Hibernate	31/10/2012	0.3
Bruno Marquete	Configuração do Hibernate	01/11/2012	0.4
Bruno Marquete	Instalação e configuração da ferrramenta de logs	01/11/2012	0.5
Bruno Marquete	Conclusão		0.6
Bruno Marquete	Ajustes finais do documento	01/11/2012	1.0

Sumário

1 Introdução

Este manual tem por objetivo abordar de forma simplificada a instalação e configuração do Hibernate aplicado ao projeto SGB (Sistema de Gestão Bibliográfica). Foge, portanto, de seu escopo trazer uma explanação profunda e detalhada de ORM (Mapeamento Objeto-Relacional), bem como do framework e suas diversas potencialidades. Atentaremos à como se dá o seu processo de configuração e de ferramentas a ele adjacentes.

2 Hibernate

Conforme definido em pesquisa realizada junto à equipe do projeto, o SGB fará uso do Hibernate como framework de ORM, em sua versão 4.1.8 (http://sourceforge.net/projects/hibernate/files/hibernate4/4.1.8.Final/). Deste modo, o projeto contará com a produtividade dada pelo mapeamento de objetos do cenário orientado a objetos para entidades do cenário relacional. Neste processo, o Hibernate será configurado para conetar-se ao banco de dados MySQL em sua versão 5.5.x.

2.1 Instalação e configuração do Hibernate

A instalação e configuração do Hibernate se dá essencialmente pela instalação das bibliotecas necessárias e pela preparação do XML que conterá as informações para

conexão ao banco de dados escolhido, neste caso o MySQL. Tais passos são descritos a seguir.

2.1.1 Instalação

No projeto SGB optou-se por utilizar o Apache Maven (http://maven.apache.org/) como ferramenta de Builder Manager. Através deste torna-se desnecessário baixar e instalar manualmente as bibliotecas utilizadas no projeto – as do Hibernate, por exemplo -, pois ele se responsabiliza por baixá-las e referenciá-las adequadamente, inclusive versões especificamente definidas.

Portanto, o única tarefa a se fazer de modo que as bibliotecas do Hibernate estejam adicionadas ao projeto é mapeá-las no arquivo XML que o Maven utiliza para se informar das bibliotecas necessárias, denominado *pom.xml*. Neste sentido, o pom.xml deve conter a seguinte declaração dentro de sua cláusula <*dependencies*>:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.1.7.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.1.7.Final</version>
</dependency>
```

De forma análoga, a cláusula <*dependencies*> deve apresentar a seguinte declaração:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.21</version>
</dependency>
```

Desta forma, será, támbém, informado ao Maven a necessidade de baixar e referenciar no projeto o driver de conexão ao banco de dados MySQL, componente vital para que o Hibernate trabalhe.

2.1.2 Configuração

De forma similar à configuração das bibliotecas a serem importadas pelo Maven, a configuração do banco de dados a ser utilizado por implementações da JPA (Java

Persistence API) - e Hibernate é um exemplo - é realizada por um arquivo XML, o *persistence.xml*. Tal arquivo deve ser criado na pasta *META-INF*, e esta deve estar posicionada no classpath da aplicação.

O *persistence.xml* a ser utilizado no projeto SGB – e, portanto, apto a conetar-se ao MySQL - deve ser algo semelhante ao seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
        xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
        version="1.0">
 <persistence-unit name="<PERSISTENCE UNIT NAME>">
      <properties>
        <property name="hibernate.ejb.cfgfile" value="/hibernate.cfg.xml"/>
        <property name="hibernate.hbm2ddl.auto" value="create"/>
       <property name="hibernate.archive.autodetection" value="class, hbm"/>
       <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
       <property name="hibernate.connection.url"
                        value="jdbc:mysql://<IP DO HOST>/<NOME BD>"/>
     <property name="hibernate.connection.username" value="<USUÁRIO>"/>
     <property name="hibernate.connection.password" value="<SENHA>"/>
     <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
     <property name="hibernate.c3p0.min_size" value="5"/>
     <property name="hibernate.c3p0.max_size" value="20"/>
     <property name="hibernate.c3p0.timeout" value="300"/>
     <property name="hibernate.c3p0.max_statements" value="50"/>
      <property name="hibernate.c3p0.idle_test_period" value="3000"/>
   </properties>
   <!-- Mapping files will go here.... →
   <class>pacote1.Entidade1</class>
```

```
    <class>pacote1.Entidade2</class>
</persistence-unit>
</persistence>
```

Como dito, o *persistence.xml* é o XML responsável por configurar a conexão disponibiblizado pelo pedrão JPA. Isso significa que, caso a implementação da JPA do projeto mude (para TopLink, por exemplo), este continuará válido. Todavia, a configuração da conexão pode, também, ser realizada no arquivo *hibernate.cfg.xml*. O uso do *persistence.xml* dispensa o uso deste, conquanto esta é uma decisão de projeto a ser realizada dado que seu uso implica na utilização ou não de classes específicas do Hibernate (HibernateSessionFactory, por exemplo). Um exemplar da estrutura do *hibernate.cfg.xml*. para conexão com MySQL é apresentado a seguir:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>

        <property name="connection.url"><IP DO HOST>/<NOME BD></property>
        <property name="connection.username"><USUÁRIO></property>
        <property name="connection.password"><SENHA></property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">create</property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>
    <property name="current_session_context_class">thread</property>
```

```
    <!-- Mapping files will go here.... →
    <mapping resource="pacote1.Entidade1" />
    <mapping resource="pacote1.Entidade2" />

  </session-factory>
</hibernate-configuration>
```

Em ambos os casos, os campos em negrito correspondem, respectivamente, à url de conexão ao banco de dados MySQL, ao usuário e à senha previamente criados para acessá-lo.

Um ponto importante a ressaltar é o valor das propriedades *hibernate.hbm2ddl.auto* e *hbm2ddl.auto.* Estas determinam como a JPA e o Hibernate, respectivamente, irão lidar com a geração automática de tabelas e campos a partir das classes de entidades codificadas – estas determinam, portanto, o esquema do banco de dados. O *hibernate.hbm2ddl.auto* pode apresentar um dos seguintes valores:

- **validate**: valida .o esquema, contudo não realiza alterações nos dados.
- **update**: atualiza o esquema.
- **create**: cria o esquema.
- **create-drop**: remove o esquema e cria-o novamente, apagando toda a última sessão.

No *persistence.xml* e no *hibernate.cfg.xml* temos os trecho ondes são declarados as entidades utilizadas, isto é, as classes que representam as tabelas do banco de dados. Nestes locais – destacados pelo comentário *"Mapping files will go here"*. - deve-se declarar todas as classes de entidades a serem utilizadas.

2.2 Instalação e configuração da ferrramenta de logs

Para ter um controle maior da persistência é preciso saber o que o Hibernate está fazendo, por exemplo, como os mapeamentos estão sendo realizados. O Hibernate disponibiliza estas informações por meio de logs. Como existem várias opções de loggers o hibernate "terceiriza" esta tarefa com uma abstração chamada Simple Logging Facade for Java (SLF4J) permitindo que o desenvolvedor utilize a implementação de logger que for mais conveniente ou mesmo criar uma própria.

No projeto SGB sugere-se a utilização do *Log4j (http://logging.apache.org/log4j/2.x/),* facilmente obtido pelo Maven através da seguinte declaração dentro da cláusula *<dependencies>:*

```
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
</dependency>
```

O log4j precisa ser configurado para que a saída de log seja direcionada para o lugar correto, o formato seja conveniente, visibilidade dos níveis de mensagens possam ser visualizados apropriadamente, etc. Todas estas propriedades do log4j são definidas no arquivo *log4j.properties*, que deve ser colocado na classpath do projeto.

Podemos, então, configurar o *log4j.properties* da seguinte forma de modo que os logs apareçam no console da IDE utilizado (no nosso caso, NetBeans):

log4j.rootCategory=INFO, CONSOLE

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender

log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout

log4j.appender.CONSOLE.layout.ConversionPattern=%r [%t] %-5p %c - %m%n

Outro aspecto a ser configurado de modo a facilitar o *debug* da aplicação é declarar o trecho a seguir no arquivo *persistence.xml*:

<property name="hibernate.show_sql" value="true"/>

Isso fará com que o Hibernate explicite ao programador todas as chamadas SQL por ele realizadas, tornando mais fácil a identificação de eventuais falhas de mapeamento e de realização de *"CRUDs"* no banco de dados.

3 Conclusão

Por meio deste sucinto manual de instalação e configuração do Hibernate espera-se que a equipe técnica do projeto SGB tenha à disposição uma referência de como será realizada a "ponte" entre o código-fonte do software a ser desenvolvido e o banco de dados, bem como entenda como utilizar ferramentas instrínsecas ao framework de ORM – Maven e Log4j, por exemplo. Deste modo, procura-se acelerar a absorção de conhecimentos necessários para realizar atividades de caráter técnico do projeto.