Hibernate ORM

O Hibernate ORM é um framework que permite que desenvolvedores codifiquem, mais facilmente, aplicações em que seus dados sobrevivam além do próprio processo da aplicação. É um framework ORM, ou seja, possui foco na persistência de dados em bancos relacionais, voltado para a realização de mapeamento objeto/relacional.

O framework possui uma arquitetura em camadas, que ajuda o usuário a utilizá-lo sem ter total conhecimento das APIs. Faz o uso do banco de dados e de dados de configuração para prover serviços de persistência para a aplicação. Desta forma, em sua estrutura, este se localiza entre os objetos Java tradicionais e o servidor do banco de dados para lidar com a persistência desses objetos, tudo baseado nos mecanismos e padrões dos objetos relacionais.

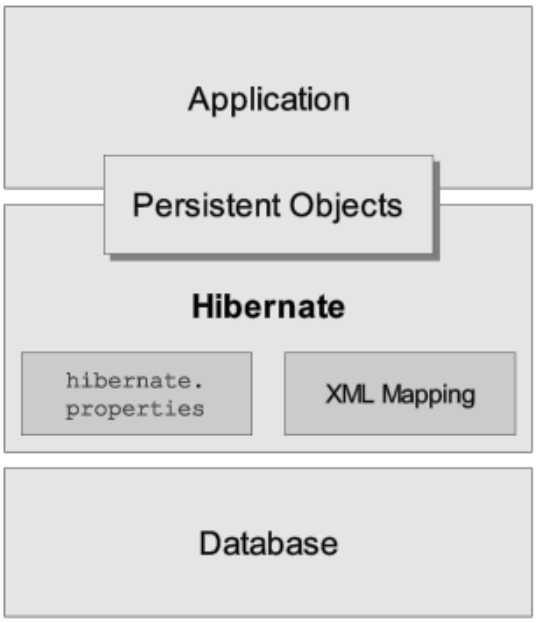

Figura 1. Visão de alto nível da arquitetura do ORM com Hibernate.

Como a própria documentação do framework e relatos de usuários indicam, o Hibernate ORM é altamente configurável e extensível. Dentro dos artefatos gerados sobre o framework há o guia de integração, em que é apresentado a possibilidade de extensão inclusive visando a integração com os servidores Java EE, com o framework Spring e

soluções caching, como Ehcache e Hazelcast. Para isso, permite o desenvolvimento de classes de persistência, em linguagens orientadas a objetos, incluindo herança, polimorfismo, associação, composição e com o *framework Java Collection*, não necessitando de interfaces ou classes base para classes de persistência, além de permitir que qualquer classe ou estrutura de dados possam ser persistidos.

O Hibernate se aproxima das especificações do modelo de componentes *Enterprise JavaBeans* (EJB), da Sun Microsystems. Este é um componente da plataforma JEE que roda em um *container* de um servidor de aplicação, tendo como objetivo fornecer um desenvolvimento rápido e simplificado de aplicações Java, com base em componentes distribuídos, transacionais, seguros e portáveis.

O framework Hibernate é um software livre de código aberto distribuído licenciado sob o LGPL 2.1 (*Lesser General Public License*) e suficientemente flexíveis para permitir o uso do Hibernate em projetos de código aberto e comercial. Liderado por Gavin King, o projeto do Hibernate foi desenvolvido por vários desenvolvedores Java espalhados pelo mundo e, posteriormente, a empresa JBoss Inc (comprada pela Red Hat) contratou os principais desenvolvedores do projeto para fazer seu suporte. Além das próprias releases, guias, *roadmap* e demais informações acerca do framework, este possui uma documentação, para todos seus releases, estando estas disponíveis em seu site (http://hibernate.org/orm /documentation/).

Para o Hibernate ORM, componentes de software são definidos como pacotes Java independentes e com funções específicas. Cada componente tem seu próprio arquivo de compilação (utilizando o *Gradle*) bem como o seu código fonte organizado em subpastas dentro do framework. Sempre que um componente precisa ser utilizado, é feito a importação do pacote do componente via código.

A arquitetura do Hibernate pode ser representada de forma geral pela ilustração abaixo:

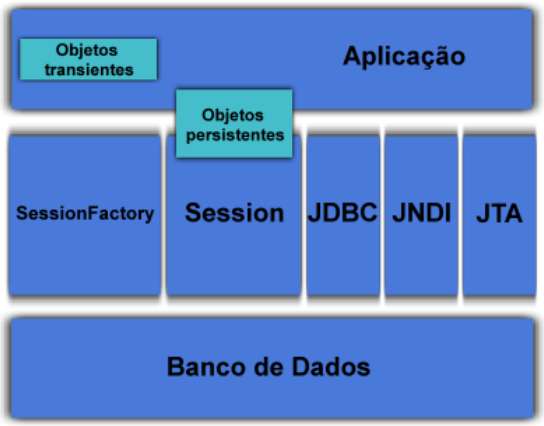

Figura 2. Abordagem simples da arquitetura do framework Hibernate.

O Hibernate ORM define um pacote chamado Core, que é o provedor da infraestrutura dos componentes, servindo como base de comunicação e utilização dos demais componentes do framework. Porém, para prover a customização do framework ou utilização de funcionalidades que não sejam as padrões, é necessário a importação dos componentes específicos do framework. Isso é feito através da utilização de interfaces públicas, que são disponibilizadas por cada componente do framework.

O Hibernate ORM tem como padrão a definição de interfaces pelos componentes que abstraem o funcionamento interno dos componentes, mas além de interfaces, o Hibernate ORM utiliza o *Domain Model Pattern* quando o componente incorpora tanto dados como comportamentos relacionados aos dados.

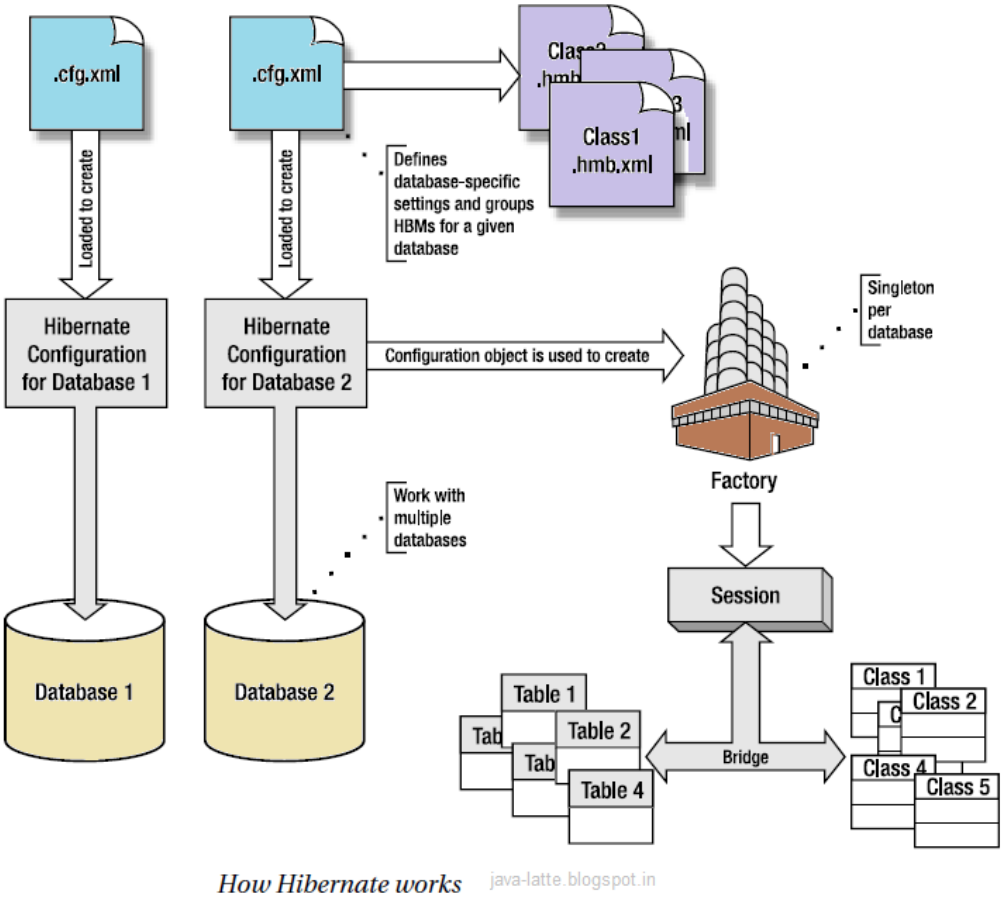

Figura 3. Visão geral do funcionamento do Hibernate.

O *Factory Pattern* é outra estratégia que o Hibernate ORM utiliza para orquestrar a criação de objetos por componentes que possuem contextos semelhantes, mas que instanciam objetos de forma diferenciada. Um exemplo mais tangível é o seu uso nos componentes de criação de query para banco de dados distintos.

Outro importante padrão utilizado pelo Hibernate ORM é o *Proxy Pattern* que é utilizado para a comunicação em componentes distribuídos por rede, principalmente para realizar consultas *lazy* entre os componentes do framework ou serviços externos (neste caso, os bancos de dados).

Por outro lado, na abordagem "completa" a aplicação não tem que lidar diretamente com JDBC/JTA e APIs, o Hibernate se encarrega de todos os detalhes, como é possível observar a seguir:

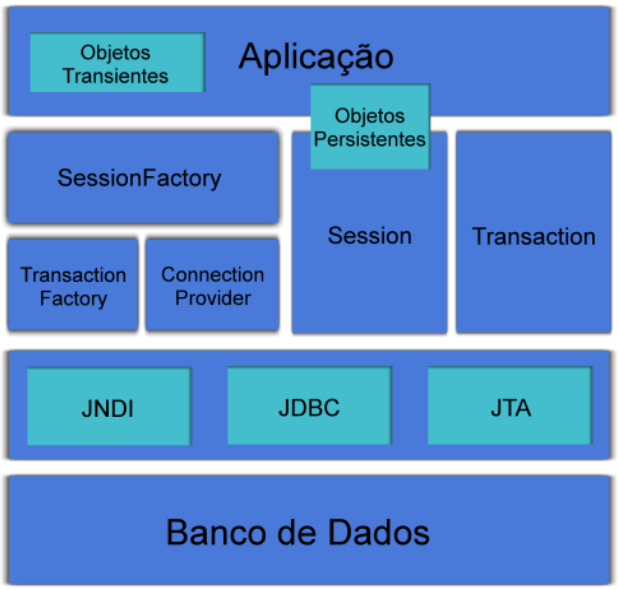

Figura 4. Abordagem completa da arquitetura do framework Hibernate.

O EJB caracteriza-se como arquitetura de componentes para o desenvolvimento e utilização de aplicações corporativas baseadas em componentes distribuídos.

SessionFactory

Uma representação de thread-safe (e imutável) do mapeamento do modelo de domínio de aplicativo para um banco de dados. Atua como uma fábrica e fornece ao aplicativo, instâncias da interface Session. Realiza ainda o armazenamento de instruções SQL e metadados de mapeamento utilizados pelo Hibernate em tempo de execução.

Um SessionFactory é muito caro para criar, portanto, para qualquer banco de dados, o aplicativo deve ter apenas um associado SessionFactory. O SessionFactory mantém serviços que o Hibernate usa em todos Session(s)os tipos de caches de segundo nível, pools de conexão, integrações de sistema de transação, etc.

Session

Principal interface usada pelos aplicativos do Hibernate. É onde o Hibernate é iniciado para a realização de operações CRUD, execução de consultas, controle de transações etc. Essa interface é responsável pelo gerenciamento de estados dos objetos.

Nos bastidores, o Hibernate Session envolve um JDBC Connection e mantém um contexto de persistência geralmente "leitura repetida" (cache de primeiro nível) do modelo de domínio do aplicativo.

Transaction

Um objeto de vida curta, de thread único, usado pelo aplicativo para demarcar limites de transação física individuais. Realiza a tarefa de abstrair o código do aplicativo do usuário que implementaria a transação subjacente. Tal abstração permite o controle dos limites de transações pelo aplicativo, mantendo a portabilidade dos aplicativos do Hibernate mesmo entre ambientes de execução distintos.

ConnectionProvider

O ConnectionProvider é opcional. Trata-se de uma fábrica (e combinações) de conexões JDBC. Este objeto abstrai a aplicação de lidar diretamente com Datasource (armazena informações sobre a conexão com a base de dados) ou DriverManager (gerenciador de *drivers*). Este objeto não é exposto para a aplicação, mas o programador pode implementá-lo ou estendê-lo.

TransactionFactory

Entendido como uma fábrica para instâncias de *Transaction*. Não é exposta à aplicação, mas como o ConnectionProvider pode ser estendido ou implementado pelo programador, também de forma opcional.

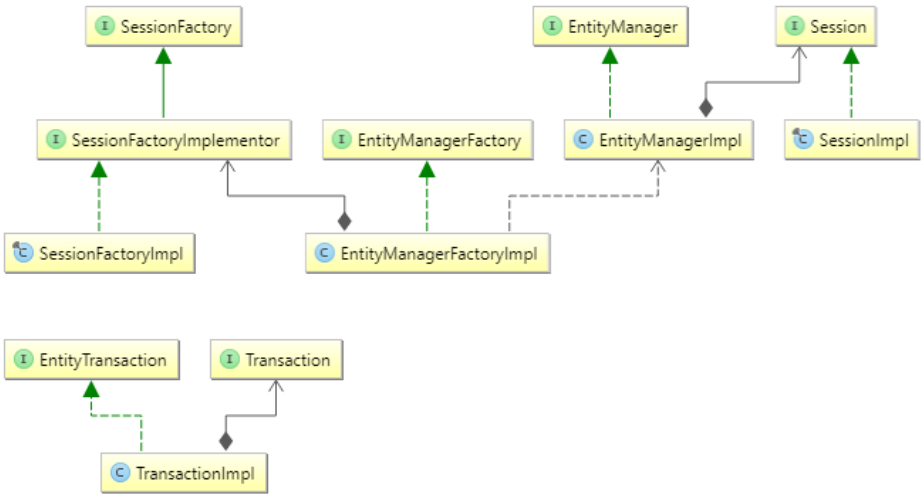

Figura 5. Diagrama de comunicação entre componentes.