Transcrição

Já discutimos um pouco sobre o JDBC e os principais problemas que Devs passaram a identificar conforme desenvolviam as aplicações. A existência desses problemas foi, justamente, o que motivou as pessoas a buscarem alternativas, tecnologias que fossem mais simples para fazer a ponte, a ligação com o banco de dados.

Dentre essas tecnologias, surgiu uma biblioteca que ficou bem famosa, chamada **Hibernate**. O Hibernate foi lançado em 2001 e criado por Gavin King, que tinha como ideia, justamente, tentar simplificar o JDBC. Supondo que temos uma aplicação web, desktop, é precisamos fazer o acesso ao banco de dados, mas não gostamos muito do modelo do JDBC, porque o código fica muito verboso, complexo, muito acoplado ao banco de dados.

Gavin King começou a pensar em uma maneira de simplificar o código e criou essa biblioteca (Hibernate) e em 2001 fez o lançamento. Mas, se percebermos, ela não tem nada a ver com o Java, ela é uma biblioteca que surgiu no mercado e que ele distribuiu gratuitamente. Assim nasceu o Hibernate, uma biblioteca famosa por fazer persistência no banco de dados como **alternativa ao JDBC e ao EJB 2**.

Na época existia a versão 2.0 do EJB 2, e era uma tecnologia bem complicada de se trabalhar. A ideia do EJB era simplificar o acesso remoto, para ter uma aplicação distribuída (cliente - servidor) e também simplificar alguns detalhes de infraestrutura, como controle transacional, segurança e outros.

Também havia a parte de persistência junto da EJB, mas ela utilizava a JDBC, era um modelo um pouco mais complexo e que favorecia muito a remotabilidade, então, qualquer chamada que se fazia era uma chamada remota, isso tinha um custo de

performance. Enfim, vários problemas surgiram e vários padrões também foram criados para resolver esse problema da EJB 2 na parte de persistência. Isso também motivou o Gavin King a criar o Hibernate.

Posteriormente, **Gavin King foi contratado pela Red Hat**, para continuar os trabalhos no Hibernate. Portanto, o Hibernate é uma teconologia que pertence à JBoss (Red Hat). O Gavin King trabalhou na JBoss e lá deu continuidade ao Hibernate e a outros projetos.

Conforme o tempo passou, foram surgindo novas versões do Hibernate, ele ficou famoso no mundo inteiro, todas as pessoas que trabalhavam com Java queriam utilizar Hibernate nos projetos para não ter que usar a EJB 2 e JDBC. Enfim, virou um projeto mega popular, e foi evoluindo, foram surgindo versões posteriores com novos recursos.

Como se tratava de uma biblioteca do mercado, surgiram também concorrentes. Portanto, estamos falando de uma biblioteca que está famosa, popular, logo, outras bibliotecas passaram a copiar, mas fazendo de outra forma.

Isso gerou um velho problema: imagine que, como Devs, estamos usando uma biblioteca, queremos trocar para outra (uma biblioteca de mercado). Para fazer isso, não é só trocar os JARs, as dependências do projeto, isso causará um impacto considerável no código.

Para todo o código em que estávamos usando o Hibernate, será necessário trocar para essa nova biblioteca, já que são outras classes, outros *imports*. Se o nosso projeto fosse grande, complexo, pensaríamos duas vezes antes de trocar. A Sun, a Oracle, o Java, não gostam disso, porque significa estar preso a um fornecedor.

Posteriormente, uma padronização da biblioteca foi criada, do modelo de persistência, que ficou conhecida como **ORM** (Object Relational Mapping) em Java, com a intenção de fazer o mapeamento, a ponte entre o mundo da "orientação a Objetos" com o "relacional" do banco de dados.

Existem esses dois mundos distintos, e a classe DAO ficava um tanto complexa, porque estávamos fazendo essa ponte entre o mundo "orientação a objetos" e mundo orientado ao "modelo relacional" do banco de dados. O Java criou uma especificação

chamada de **JPA**, Java Persistence API, que é a especificação para padronizar o mapeamento a objeto relacional no mundo Java.

Com a JPA, criou-se um padrão para que não ficássemos reféns de uma biblioteca. Os *frameworks*, as bibliotecas, começaram a implementar a JPA. No código, ao invés de fazer os *imports* das classes e interfaces do Hibernate, passou-se a fazer a da JPA, que é a especificação.

Portanto, a biblioteca se tornava uma implementação, e, para trocar de implementação, só precisávamos trocar os JARs, uma ou outra configuração, mas o código, em si, continuava intacto, inalterado, por não depender de uma implementação.

A JPA só foi **lançada em 2006**, então, de 2001 a 2006, precisávamos utilizar o Hibernate ou os seus concorrentes sem a abstração da especificação. O Hibernate foi evoluindo, foram surgindo novos recursos e depois isso foi incorporado na **versão 2.0 da JPA, lançada em 2009**.

O **Hibernate, em 2010**, um ano depois, lançou a **versão 3.5.0**, que era **compatível com a JPA 2.0**. Portanto, se quiséssemos usar a JPA 2.0, podíamos utilizar o Hibernate como implementação. Se posteriormente quiséssemos trocar por outras implementações, seria fácil, não causaria impacto no projeto inteiro. Essa é a grande vantagem de se utilizar uma especificação ao invés de usar diretamente uma implementação e ficar preso a ela.

No mercado, ficamos com algo parecido com esse diagrama, em que uma seta sai de cada um dos três retângulos (1. Hibernate, 2. EclipseLink, 3. OpenJPA) e aponta para o retângulo que está acima deles: JPA. Ou seja, temos em cima a JPA, que é a especificação, e temos também várias implementações, como o Hibernate, o EclipseLink, OpenJPA, dentre outras implementações. Essas são as três principais da JPA.

Para trabalharmos com a JPA, temos que escolher uma dessas implementações. Isto é, não se usa a JPA "pura", porque ela é só a "casca", a abstração. Nós precisamos de alguém que implemente os detalhes, quem faz esse trabalho são bibliotecas como o Hibernate.

Como o Hibernate foi a primeira biblioteca, ele foi quem começou esse movimento, por isso acabou se tornando a biblioteca padrão, a mais utilizada, mais popular como implementação da JPA. Mas, nada nos impede de usar outras implementações. Se todas estão seguindo a JPA, todas precisam atender ao que está descrito na especificação da JPA.

Além disso, podemos trocar uma implementação pela outra, e tudo que estiver na especificação vai funcionar. Só teremos problemas se estivermos utilizando algo que é específico da implementação. Às vezes, o Hibernate costuma fazer isso, segue a JPA, mas tem alguns recursos que são específicos dele.

Se utilizarmos, está tudo certo, estamos ganhando esse novo recurso. Mas, o lado negativo, já que esse recurso é específico do Habernate. Então, por exemplo, se quisermos trocar o Hibernate pelo EclipseLink, perderemos essa funcionalidade. Assim, temos que tomar esse cuidado e avaliar se vale mesmo a pena essa dependência.

O EclipseLink é a implementação de referência da JPA. Sempre que surge uma nova versão da JPA, o EclipseLink já está implementando, pois, é nele que são feitos os testes e ele sai com a nova versão da JPA. Ele é a implementação de referência, porém, o Hibernate é a principal, a mais popular no mercado. Por isso, neste curso trabalharemos com o Hibernate como implementação da JPA.

Fechamos essa parte de motivação. Nosso objetivo era apenas discutir um pouco e entender o que é a JPA, porque ela foi criada, de que nasceu, o que é o Hibernate, qual a diferença de Hibernate e JPA. Agora que alinhamos esses conhecimentos, no próximo vídeo, partiremos para a prática. Começaremos a aprender como utilizar a JPA e Hibernate no projeto.

Vejo vocês lá!! Abraços!!