01

Habilitando Cache de Segundo Nível

Transcrição

Nosso propósito aqui é fazer com que a homepage da Casa do Código seja acessada mais rapidamente a partir do segundo usuário. Toda vez que acessamos a home, o console carrega vários livros do banco de dados, ou seja, a aplicação pede as informações para o banco e as guarda em sua memória.

Para que esse processo seja mais rápido, teremos que pegar tais informações do banco na primeira vez que o usuário acessar, já na segunda vez os dados estarão na memória da aplicação para os outros usuários. O acesso na memória é muito mais rápido. Logo, o que temos que fazer é implementar o código para que isso aconteça, é o que chamamos de cache. Como está agora, cada acesso na aplicação faz 2 SELECTS.

Vamos lá. Em `LivroDao.java` temos dois métodos, o `ultimosLancamentos()` e o `demaisLivros()`. Podemos fazer com que as queries vão para o cache, usando o `setHint()`:

```
public List<Livro> ultimosLancamentos() {
    String jpql = "select l from Livro l order by l.id desc";
    return manager.createQuery(jpql, Livro.class)
    .setMaxResults(5)
    .setHint(QueryHintsHINT_CACHEABLE, true).getResultList();
}
```

Lembrando de fazer também para o `demaisLivros()`.

O JPA é o responsável por pegar os dados na nossa aplicação, mas a especificação é o Hibernate e ele precisa saber que queremos fazer cache desses dados na memória da aplicação. Por isso pedimos para que o resultado da query seja mantido. A Classe "QueryHints" não foi carregada, devemos então fazer isso no `pom.xml` colocando mais algumas dependências:

```
<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>1.0.0.Draft-16</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>javax.transaction-api,/
    <artifactId>
    <version>1.2</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.10.Final</version>
    <scope>provided</scope>
</dependency>
```

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.10.Final</version>
    <scope>provided</scope>
</dependency>
```

Todas as APIs que iremos carregar em cima das dependências já as temos de fato na aplicação porque o Wildfly já faz isso.

Fazendo um teste, ainda vemos duas queries por acesso, o que queremos será sempre duas queries não importando quantos acessos existam. O Hibernate precisa que digamos explicitamente que queremos fazer cache com as queries.

Vamos abrir o `persistence.xml`, que é onde fazemos as configurações de cache, e falar para o Hibernate que queremos fazer cache dessas queries, dentro de uma *property*:

```
<properties>
    <property name="hibernate.cache.use_query_cache" value="true"/>
    //...
</property>
```

Dessa forma usaremos o cache para query. Mas o JPA também precisa saber que queremos cachear, então usamos a opção `shared-cache-mode` habilitando seletivamente:

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
```

Subindo o servidor, vejamos o resultado da aplicação. De novo, um monte de queries! Perceba, no console, que está buscando o id:

```
where
    livro0_.id=?
```

Treze (13) queries foram feitas pelo id, ou seja, um querie para cada livro. Parece que pioramos ainda mais a situação. Isso acontece porque toda vez que habilitamos o cache, o resultado da query vai ficar cacheado, mas não falamos que a **entidade** é que tem que ser cacheada, guardando apenas o id dela.

Para resolver esse problema precisamos falar que as entidades serão cacheadas, ou seja, os Livros. Logo, abrimos o `Livro.java` e anotamos a Classe "Livro" com `@Cacheable`:

```
import javax.persistence.Cacheable;

//...

@Entity
@Cacheable
public class Livro {
    //...
}
```

Hoje em dia o JPA está melhor integrado com os frameworks de implementação de especificações e com algumas funcionalidades avançadas, como o cache, o que antigamente não era tão simples.

Testando de novo, na primeira atualização, como esperado, foram feitas duas queries. Depois podemos atualizar quantas vezes quisermos a página, o console não mostrará mais nenhuma query. Agora sim está tudo na memória da aplicação, a deixando mais rápida.