04

Ainda sobre qualificadores

Transcrição

Veja a classe `LoginBean` depois de todas as alterações que fizemos:

```
@Named
@RequestScoped
public class LoginBean implements Serializable {

    private static final long serialVersionUID = 1L;

    private Usuario usuario = new Usuario();

    private UsuarioDao usuarioDao;

    private MessageHelper helper;

    private Map<String, Object> sessionMap;

    @Inject
    public LoginBean(UsuarioDao usuarioDao, MessageHelper helper, @ScopeMap(Scope.SESSION) Map<Strir
        this.usuarioDao = usuarioDao;
        this.helper = helper;
        this.sessionMap = sessionMap;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public String efetuaLogin() {
        System.out.println("fazendo login do usuario " + this.usuario.getEmail());

        boolean existe = usuarioDao.existe(this.usuario);
        if(existe) {
            sessionMap.put("usuarioLogado", this.usuario);
            return "livro?faces-redirect=true";
        }

        helper
            .onFlash()
            .addMessage(new FacesMessage("Usuário não encontrado"));

        return "login?faces-redirect=true";
    }

    public String deslogar() {
        sessionMap.remove("usuarioLogado");
        return "login?faces-redirect=true";


    }
}
```

A classe está bem mais coesa: recebe todas as dependências injetadas e simplesmente utiliza as dependências nos métodos da classe. E das dependências, apenas uma é do projeto `livraria` : o `UsuarioDao` . As outras dependências vem da nossa biblioteca.

Vamos olhar a classe `UsuarioDao` e ver se temos algum ponto que podemos melhorar:

```
public class UsuarioDao {

    private EntityManager em;

    @Inject
    public UsuarioDao(EntityManager em) {
        this.em = em;
    }

    public boolean existe(Usuario usuario) {
        TypedQuery<Usuario> query = em.createQuery(
                  " select u from Usuario u "
                + " where u.email = :pEmail and u.senha = :pSenha", Usuario.class);

        query.setParameter("pEmail", usuario.getEmail());
        query.setParameter("pSenha", usuario.getSenha());
        try {
            query.getSingleResult();
        } catch (NoResultException ex) {
            return false;
        }

        em.close();

        return true;
    }

}
```

O método `existe()` cria uma `TypedQuery` , e portanto é uma dependência do método. Vamos resolver a dependência utilizando CDI. Queremos receber de algum forma, o `TypedQuery` :

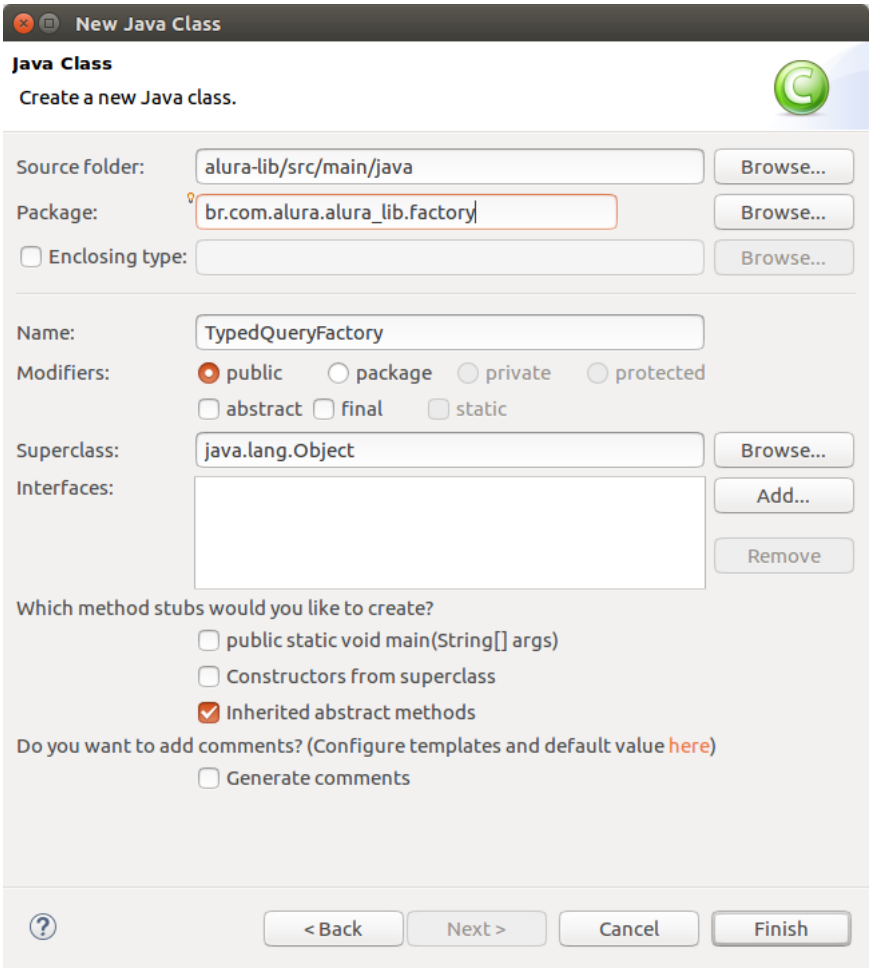
```
public class UsuarioDao {

    @Inject
    private EntityManager em;

    @Inject
    private TypedQuery<Usuario> query;

    // ...
}
```

O `TypedQuery` é uma interface, que estende da interface `Query` e possui um tipo genérico:

```
public interface TypedQuery<X> extends Query {
```

Por ser uma interface, o CDI não irá conseguir instanciar e injetar diretamente. Será preciso que alguém produza essa `TypedQuery`. Na biblioteca, vamos criar uma classe que irá produzir o `TypedQuery`.

A classe ficará no pacote `br.com.alura.alura_lib.factory` com o nome `TypedQueryFactory`:



Para produzir o objeto, precisamos verificar como ele está sendo produzido no `UsuarioDao`:

```
TypedQuery<Usuario> query = em.createQuery(
                  " select u from Usuario u "
                + " where u.email = :pEmail and u.senha = :pSenha", Usuario.class);
```

Precisamos de um `EntityManager`, de uma `String` com a *query* e também passamos a classe que referencia a *query*. Basicamente precisamos das seguintes informações:

```
public class TypedQueryFactory {

    @Inject
    private EntityManager em;

    public <X> TypedQuery<X> factory() {

        Class classe;
        String jpql;
```

```
        return em.createQuery(jpql, classe);
    }
}
```

Já vimos como obter a classe por meio do `Injection Point`:

```
@Produces
public <X> TypedQuery<X> factory(InjectionPoint point) {

    ParameterizedType type = (ParameterizedType) point.getType();

    @SuppressWarnings("unchecked")
    Class classe = (Class<X>) type.getActualTypeArguments()[0];

    String jpql;
    return em.createQuery(jpql, classe);
}
```

Falta resolvermos apenas a JPQL... Queremos passá-la quando declararmos a dependência. Algo semelhante a isso:

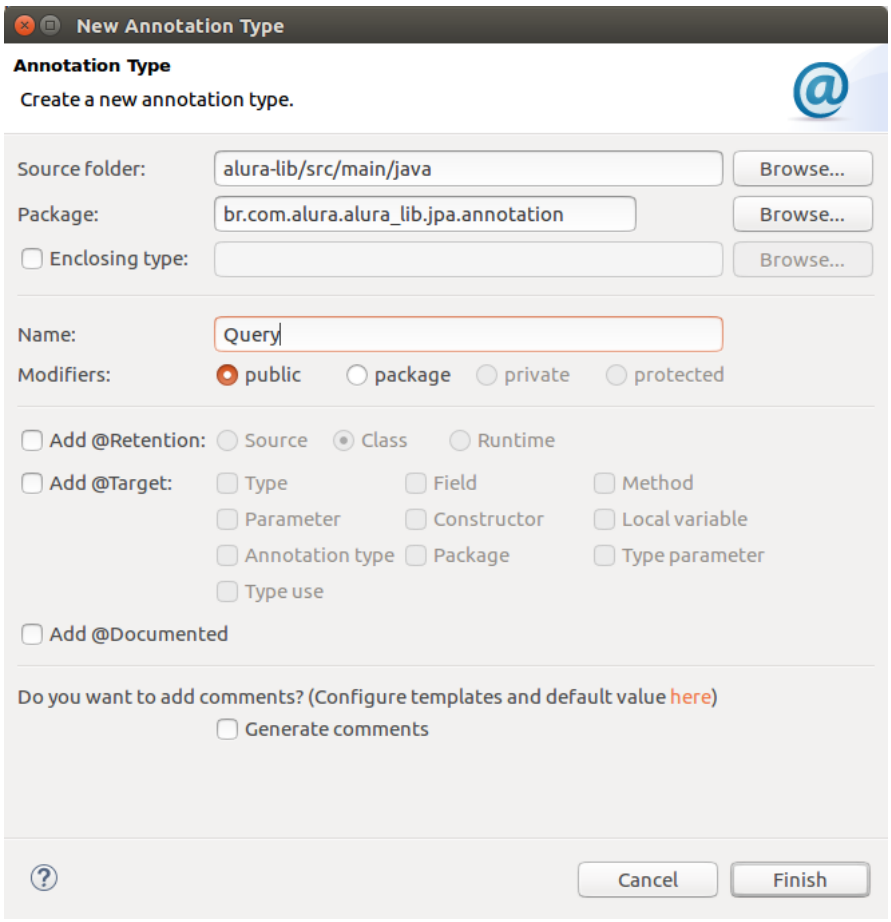
```
@Inject("select u from Usuario u where u.email = :pEmail and u.senha = :pSenha")
private TypedQuery<Usuario> query;
```

Vamos aproveitar e remover a criação do `TypedQuery` do método `existe()`:

```
public boolean existe(Usuario usuario) {

    query.setParameter("pEmail", usuario.getEmail());
    query.setParameter("pSenha", usuario.getSenha());
    try {
        query.getSingleResult();
    } catch (NoResultException ex) {
        return false;
    }

    em.close();

    return true;
}
```

Mas com a anotação `@Inject` não conseguimos passar a *query* pra o método. A `String` com a *query* é o que vai diferenciar o `TypeQuery<Usuario>` de um `TypeQuery` de outro tipo. Já que queremos diferenciar, precisamos de um qualificador.

Vamos criar uma nova *annotation*, no pacote `br.com.alura.alura_lib.jpa.annotation`, chamada `Query`.



Já vamos utilizar a anotação no produtor, passando a *query*. Dessa forma quando alguém utilizar o qualificador com essa *query*, vamos receber o `TypedQuery` correspondente.

```
@Produces
@Query("select u from Usuario u where u.email = :pEmail and u.senha = :pSenha")
public <X> TypedQuery<X> factory(InjectionPoint point) {
```

A anotação ficará com o seguinte código:

```
@Qualifier
@Target({ElementType.PARAMETER, ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Query {

    String value();
}
```

O que precisamos fazer agora é pegar o valor que foi passado para a anotação. Pois esse valor é a *query*. Para isso vamos recorrer ao `InjectionPoint`.

Obtemos a anotação através da chamada `getAnnotated().getAnnotation(Query.class)`. A partir disso temos acesso aos atributos, nesse caso pegamos o `value()`.

```
String jpql = point.getAnnotated().getAnnotation(Query.class).value();
```

O ponto é que agora que criamos o qualificador, para cada *query* que temos iremos precisar de um produtor. Vimos que quando criamos um qualificador, todos os atributos que temos dentro do qualificador são utilizados para resolver o conflito. Só que o atributo que estamos passando, uma `String`, não é um atributo relevante para ele distinguir quais são os `bean`s que devem ser injetados.

O CDI tem uma forma para indicarmos que um atributo não deve ser levado em consideração no momento de distinguir um *bean* de outro. Para isso anotamos o atributo com `@Nobinding`:

```
public @interface Query {

    @Nonbinding String value();
}
```

Dessa forma, o atributo deixará de ser avaliado para distinguir um *bean* de outro. No produtor, não precisamos mais passar a *query*. Mas como não definimos um valor *default* para o `value()`, vamos passar uma `String` vazia:

```
@Produces
@Query("")
public <X> TypedQuery<X> factory(InjectionPoint point) {
```

Vamos instalar o `alura-lib` no repositório local e em seguida clicar com o botão direito no projeto `livraria` e escolher as opções "Maven > Update Project".

Na classe `UsuarioDao`, podemos utilizar a anotação:

```
@Query("select u from Usuario u where u.email = :pEmail and u.senha = :pSenha")
private TypedQuery<Usuario> query;
```

Deixamos de utilizar o `EntityManager` no nosso `UsuarioDao`. Agora precisamos apenas da *query* pra executar:

```
public class UsuarioDao {

    @Inject
    @Query("select u from Usuario u where u.email = :pEmail and u.senha = :pSenha")
    private TypedQuery<Usuario> query;

    public boolean existe(Usuario usuario) {

        query.setParameter("pEmail", usuario.getEmail());
        query.setParameter("pSenha", usuario.getSenha());
        try {
            query.getSingleResult();
        } catch (NoResultException ex) {
            return false;
        }

        return true;
    }

}
```

Utilize o atalho "Ctrl + Shift + O" para organizar os *imports*.

Vamos testar para ver se tudo funcionou. Dar um *clean* e iniciar o Tomcat. E tudo funciona!

Vimos que com os qualificadores podemos definir que certos atributos não serão utilizados para distinguir um *bean* de outro.

O `TypedQuery` que criamos, foi apenas para exemplificar o uso da anotação `@Nobinding` . Faz sentido o `UsuarioDao` criar sua própria *query*. Já que o DAO é a classe que faz comunicação com o banco de dados.