04

Configurações importantes do Pool C3P0

Transcrição

##Configurações importantes do Pool C3P0

Como vimos, um `Datasource` isola as informações sobre acesso ao banco de dados. Assim, ainda precisaremos manter as configurações sobre o *usuário, senha, url* e *classe do Driver*. E, portanto, vamos usar os métodos `setUser`, `setPassword`, `setDriverClass` e `setJdbcUrl`:

```
@Bean
public DataSource getDataSource() throws PropertyVetoException {
    ComboPooledDataSource dataSource = new ComboPooledDataSource();

    dataSource.setDriverClass("com.mysql.jdbc.Driver");
    dataSource.setJdbcUrl("jdbc:mysql://localhost/projeto_jpa");
    dataSource.setUser("root");
    dataSource.setPassword("");

    dataSource.setInitialPoolSize(3);
}
```

Como vimos, precisamos abrir uma certa quantidade de conexões para compartilharmos entre os clientes. Para isso, podemos usar o método `setMinPoolSize` que define um número mínimo de conexões que devem ser criadas de antemão e que estarão esperando clientes. Lembrando que não existe um número mágico para todos os casos, o ideal é escolher com base em análises de tráfego da aplicação.

Quando não houver mais conexões disponíveis para serem usadas pelos clientes, o *pool* irá criar novas conexões para atender a demanda.

```
@Bean
public DataSource getDataSource() throws PropertyVetoException {
    ComboPooledDataSource dataSource = new ComboPooledDataSource();

    dataSource.setDriverClass("com.mysql.jdbc.Driver");
    dataSource.setJdbcUrl("jdbc:mysql://localhost/projeto_jpa");
    dataSource.setUser("root");
    dataSource.setPassword("");

    dataSource.setMinPoolSize(3);

}
```

Vamos fazer um pequeno teste para verificar se o *C3P0* realmente criará *três* conexões iniciais? Para isso, vamos criar uma classe chamada `TestaPool` com um método `main`.

```
public class TestaPool {
    public static void main(String[] args) {
```

```
    }
}
```

Vamos instanciar um `ComboPooledDataSource` a partir da classe `JpaConfigurator` para criarmos nosso teste.

```
public class TestaPool {
    public static void main(String[] args) throws PropertyVetoException {
        ComboPooledDataSource dataSource = (ComboPooledDataSource) new JpaConfigurator().getDat
    }
}
```

Com o `dataSource` em mãos, vamos criar uma conexão a partir dele. Além disso, vamos logar algumas informações para analisarmos o que está acontecendo:

```
public class TestaPool {
    public static void main(String[] args) throws PropertyVetoException, SQLException {
        ComboPooledDataSource dataSource = (ComboPooledDataSource) new JpaConfigurator().getDat

        dataSource.getConnection();

        System.out.println("Conexões existentes: " + dataSource.getNumConnections());
        System.out.println("Conexões ocupadas: " + dataSource.getNumBusyConnections());
        System.out.println("Conexões ociosas: " + dataSource.getNumIdleConnections());

    }
}
```

Para fixar: após obter uma conexão com o `pool` a partir do método `getConnection`, logamos as quantidades de conexões **existentes**, **ocupadas** e **ociosas**. Após executar essa classe receberemos a seguinte saída:

```
Conexões existentes: 3
Conexões ocupadas: 1
Conexões ociosas: 2
```

Podemos ver que, da forma que configuramos, anteriormente, possuímos um número inicial de **três** conexões iniciais no pool. Porém, como obtemos uma conexão a partir do `getConnection` uma delas ficou ocupada e sobraram **duas** ociosas.

Limitando a quantidade de conexões

Da mesma forma que definimos um valor inicial para o tamanho do *pool*, podemos também definir um valor máximo de conexões a serem criadas. Lembrando que o *pool* nunca criará mais conexões do que as que foram estabelecidas. Caso não hajam mais conexões para serem usadas o cliente precisará esperar por uma conexão disponível. Para configurar o valor máximo, usamos o método `setMaxPoolSize`:

```
@Bean
public DataSource getDataSource() throws PropertyVetoException {
    ComboPooledDataSource dataSource = new ComboPooledDataSource();

    dataSource.setDriverClass("com.mysql.jdbc.Driver");
    dataSource.setJdbcUrl("jdbc:mysql://localhost/projeto_jpa");
```

```
        dataSource.setUser("root");
        dataSource.setPassword("");

        dataSource.setMinPoolSize(3);
        dataSource.setMaxPoolSize(5);

        return dataSource;
    }
```

##Testando o Pool de conexões

Vamos fazer mais um teste! Dessa vez, vamos verificar se esse `setMaxPoolSize` realmente funciona. Pra isso, vamos adicionar na nossa classe teste um loop de 10 iterações:

```
public class TestaPool {
    public static void main(String[] args) throws PropertyVetoException, SQLException {
        ComboPooledDataSource dataSource = (ComboPooledDataSource) new JpaConfigurator().getData

        for(int i = 0; i < 10; i ++) {
            dataSource.getConnection();

            System.out.println(i + " - Conexões existentes: " + dataSource.getNumConnections())
            System.out.println(i + " - Conexões ocupadas: " + dataSource.getNumBusyConnections()
            System.out.println(i + " - Conexões ociosas: " + dataSource.getNumIdleConnections()

            System.out.println("");
        }
    }
}
```

Quando executarmos essa classe, teremos a seguinte saída:

```
0 - Conexões existentes: 3
0 - Conexões ocupadas: 1
0 - Conexões ociosas: 2

1 - Conexões existentes: 3
1 - Conexões ocupadas: 2
1 - Conexões ociosas: 1

2 - Conexões existentes: 3
2 - Conexões ocupadas: 3
2 - Conexões ociosas: 0

3 - Conexões existentes: 4
3 - Conexões ocupadas: 4
3 - Conexões ociosas: 0

4 - Conexões existentes: 5
4 - Conexões ocupadas: 5
4 - Conexões ociosas: 0
```

Das **dez** interações que programamos no `for`, ele realiza apenas cinco. Por que isso acontece? Qual é o valor que configuramos no `setMaxPoolSize`? **Cinco**!!! Esse número nunca será ultrapassado, ok!?

Uma outra observação interessante é que nas três primeiras iterações tivemos o mesmo número fixo de **conexões existentes**. Porém, as quantidades de conexões **ocupadas** e **ociosas** incrementam inversamente.

Isso acontece porque o *pool* armazena inicialmente **três** conexões ociosas (sem uso). Na primeira obtenção uma delas torna-se **ocupada** e, por consequência, o número de conexões ociosas diminui. O mesmo evento ocorre na segunda e

terceira iteração.

A partir da quarta não existem mais conexões no pool, tornando necessário a criação de uma nova conexão.