Spring-Connection pools

Recap

In the previous chapter, we:

- Explored how JDBC works in Spring.
- Used JdbcTemplate, which simplifies database interactions by reducing boilerplate code.
- Configured JdbcTemplate in an XML file.

Now, we will focus on connection pools for optimizing database performance..

JDBCTemplate configuration in Spring XML

To use JdbcTemplate, we inject it into the DAO bean's constructor.

In application.xml:

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource"/>
</bean>
```

dataSource must be defined separately and injected into JdbcTemplate.

JdbcTemplate methods and constructor

The following link is useful to find the name of the package and what properties are needed:

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html

Look at the constructors which has an argument named Datasource.

So we need to create an object of type DataSource and inject it into the template.

Understanding DataSource in Java

- DataSource is part of the standard Java API.
- It provides database connection management.
- Database vendors implement this interface for efficient connection handling.

Spring provides built-in implementations, including SimpleDriverDataSource (for testing) and DBCP (for production).

Using SimpleDriverDataSource

Spring provides SimpleDriverDataSource, a basic implementation of DataSource.

It is not recommended for production, as it creates a new connection for every operation.

This is the link to this class:

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/datasource/SimpleDriverDataSource.html

SimpleDriverDataSource in Spring

There are four properties we should set for this class:

- driverClass
- url
- user name
- password

Configuring SimpleDriverDataSource in XML

```
<!-- DataSources -->
<bean id="dataSource"

class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
        <property name= "driverClass" value="org.hsqldb.jdbcDriver" />
        <property name= "url" value
                        ="jdbc:hsqldb:file:database.dat;shutdown=true" />
        <property name = "username" value="sa"/>
        <property name="password" value ="" />
</bean>
```

Injecting JdbcTemplate in the Dao bean

The template should be injected in the constructor of the Dao bean.

```
<!-- Dao beans -->
<bean id="bookDao"
class="se.yrgo.spring.data.BookDaoSpringJdbcImpl" >
    <constructor-arg ref="jdbcTemplate"/>
</bean>
```

Using Connection Pools for Production

Why Use Connection Pools?

- Opening a database connection is expensive.
- A connection pool maintains a pool of open connections.
- When a client requests a connection, it reuses an existing connection instead of opening a new one.

Using Connection Pools for Production

How It Works:

- The server starts up and initializes a pool of database connections.
- A client requests a database operation → a free connection is assigned.
- When the operation is complete, the connection is returned to the pool instead of being closed.
- The same connection can be reused for other clients, improving performance.

Connection Pool Implementations

Several libraries provide connection pooling:

- DBCP (Apache Commons)
- C3PO
- Proxool
- Tomcat Connection Pool

Among these, DBCP (Database Connection Pooling) from Apache is widely used

Using Apache DBCP for Production

1. DBCP dependency in Maven:

```
<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->
 <dependency>
       <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.4</version>
 </dependency>
```

Using Apache DBCP for Production

2. Configuring DBCP in application.xml

To switch from SimpleDriverDataSource to DBCP, update the dataSource bean:

```
<!-- Connection Pool for Production -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
    <property name="url" value="jdbc:hsqldb:file:database.dat;shutdown=true"/>
    <property name="username" value="sa"/>
    <property name="password" value=""/>
</bean>
```

Important Change: driverClassName replaces driverClass.

Comparison: SimpleDriverDataSource vs. DBCP

Feature	`SimpleDriverDataSource` (Testing)	`DBCP` (Production)
Connection Handling	Creates a new connection each time	Uses a pool of connections
Performance	Slow due to frequent connections	Faster, reuses connections
Recommended For	Testing, development	Production
Configurable Properties	Basic settings only	Supports advanced tuning

Final Configuration: Injecting Connection Pool into JdbcTemplate

Once DBCP is configured, update the JdbcTemplate bean:

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource"/>
</bean>
```

Inject JdbcTemplate into DAO Bean:

```
<bean id="bookDao" class="se.yrgo.spring.data.BookDaoSpringJdbcImpl">
    <constructor-arg ref="jdbcTemplate"/>
</bean>
```

destroy-method=”close”

To avoid locking the database, in the xml file we add destroy-method="close" to the dataSource bean. This will close the method for this bean.

```
<!-- DataSources -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
                                  destroy-method="close">
```

If we do not do that, the database will be locked and two more files will be produced.
One of them is *database.dat.lck.* This file is a lock file and it tells us that the database is still open and was not closed.

By this destroy-method="close" Spring will automatically close this method when we call the close method on the container in the main method.

init-method="createTables"

In Spring, you can specify an init-method in the <bean> configuration to invoke a particular method of the bean after the bean is fully instantiated and all the dependencies (like the JdbcTemplate have been injected). Here the method createTables is responsible for creating book table in the BookDaoSpringJdbcImpl :

```
	private void createTables() {
		try {
			jdbcTemplate.update(CREATE_TABLE_SQL);
		}catch(Exception e) {
			System.err.println("Table already exists");}}
```

This method will never be called unless in the xml file in the bookDAO, we put an init-method

```
<!-- Dao beans -->
<bean id="bookDao" class="se.yrgo.spring.data.BookDaoSpringJdbcImpl"
                        init-method="createTables">
	<constructor-arg ref="jdbcTemplate">
	</constructor-arg>
</bean>
```