Pool de conexiones

¡Comencemos!

¿Qué aprenderás?

- Entender el concepto de pool de conexiones.
- Entender las razones por la cual se implementan.
- Crear un pool de conexiones.

Introducción

En el mundo actual, las aplicaciones web dan servicio a grandes cantidades de usuarios, los cuales dependiendo de la naturaleza del software esperan en tiempo real y de la forma más fiable y rápida los datos que le den valor a su negocio u objetivo.

La alta demanda de información obliga a las aplicaciones a manejar una gran cantidad de conexiones a la base de datos en un mismo instante y de forma asíncrona. Esta problemática resuelve los pool de conexiones los cuales van entregando conexiones bajo demanda según la exigencia del momento.

Concepto de pool de conexiones

El rendimiento de una aplicación web es una de las grandes razones por la cual se implementan los pool de conexiones. Es cosa de imaginar a una aplicación de antaño (de escritorio) que estaba instalada en la máquina del cliente y tenía acceso exclusivo a una conexión de base de datos.

En esos tiempos, la cantidad de usuarios concurrentes estaban limitados por la cantidad de computadores en la cual estaban instaladas las aplicaciones (una oficina con 10 empleados por ejemplo).

Para este escenario, cada computador tiene una conexión abierta a la base de datos exclusiva para él y no se libera hasta que su trabajo no esté terminado. Ahora pensemos en una aplicación empresarial web, en la cual teóricamente podría tener toda la internet conectada en un mismo lapso de tiempo. Si se sigue con la técnica de una conexión por usuario, obviamente el rendimiento del sistema sería paupérrimo y dada la cantidad de solicitudes terminará por botar el sistema.

El pool de conexiones llega a subsanar esta problemática, proporcionando una “piscina” de conexiones listas para ser entregadas a los clientes que lo requieran de forma asíncrona y disponible la mayor parte del tiempo.

En este caso, los usuarios no esperan a que otro usuario libere la conexión permitiendo que la aplicación tenga un funcionamiento constante y disminuyendo la posibilidad de caídas.

Funcionamiento

El principal actor es el servidor de aplicaciones que tiene N conexiones previamente creadas y con perfecta conexión a la base de datos. Estas conexiones están en espera listas para ser usadas. En el momento que una aplicación pide un número determinado de conexiones, el servidor web solamente toma las que están en espera y las disponibiliza a la aplicación bajo demanda. Este procedimiento es muy rápido, ya que las conexiones ya están creadas.

Cuando la aplicación termina de usar la conexión, la misma no es cerrada ni finalizada, sino que es devuelta al pool de conexiones lista para ser usada por otra petición. En el momento en que la conexión está siendo usada otra petición en el mismo rango de tiempo podría pedir una conexión y el servidor toma otra conexión del pool y se la entrega.

Imagen 1. Ejemplificando un pool de conexiones.
Fuente: Desafío Latam.

Configuración

Para utilizar un pool de conexiones, utilizaremos el proyecto anterior. Aprovechando que la arquitectura está utilizando un patrón dao, veremos lo fácil que es implementar una nueva técnica de conexión.

Actualmente, el sistema conecta a la base de datos utilizando la clase *AdministradorConexion* la cual tiene un método de nombre *generaConexion()*.

Este método en su estructura utiliza una conexión directa mediante JDBC.

```
20    protected Connection generaConexion() {
21        String usr = "sys as sysdba";
22        String pwd = "admin";
23        String driver = "oracle.jdbc.driver.OracleDriver";
24        String url = "jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";
25
26        try {
27            Class.forName(driver);
28            conn = DriverManager.getConnection(url,usr,pwd);
29        }catch(Exception ex) {
30            ex.printStackTrace();
31        }
32        return conn;
33    }
```

Imagen 2. Generar conexión.
Fuente: Desafío Latam.

Para utilizar un pool de conexiones, implementaremos otro método protegido de nombre *generaPoolConexion()*.

```
35    protected Connection generaPoolConexion() {
36        Context initContext;
37        try {
38            initContext = new InitialContext();
39            DataSource ds = (DataSource) initContext.lookup("java:/comp/env/jdbc/ConexionOracle");
40            try {
41                conn = ds.getConnection();
42            } catch (SQLException e) {
43                e.printStackTrace();
44            }
45        } catch (NamingException e) {
46            e.printStackTrace();
47        }
48        return conn;
49    }
```

Imagen 3. Generar pool de conexiones.
Fuente: Desafío Latam.

Como se puede ver, ya no tenemos los parámetros de usuario, driver, conexiones, etc. La línea 36 muestra la instancia de la clase *Context* cuya función es generar un contexto de ejecución.

Notar que la instancia no se inicializa hasta entrar al *try catch*. Al momento de inicializar *InitialContext()* se crea otra instancia de la clase *DataSource*.

El objeto *initContext* utiliza un método de nombre *initContext.lookup* en donde se le pasa como parámetro una url. Esta url *java:/comp/env/jdbc/ConexionOracle* es el nombre que se asignó al pool de conexiones. Pero no se ven las credenciales de la base de datos ni la url de conexión como en el método con JDBC. Eso es porque la conexión se dispone a nivel de configuración como veremos a continuación.

¿Qué es un *lookup*? Es una mirada. En palabras más técnicas, mediante este método se hace una búsqueda del recurso en la ruta dispuesta como parámetro. Lo que hará es buscar tales conexiones en un archivo de configuración del servidor de aplicaciones *tomcat*.

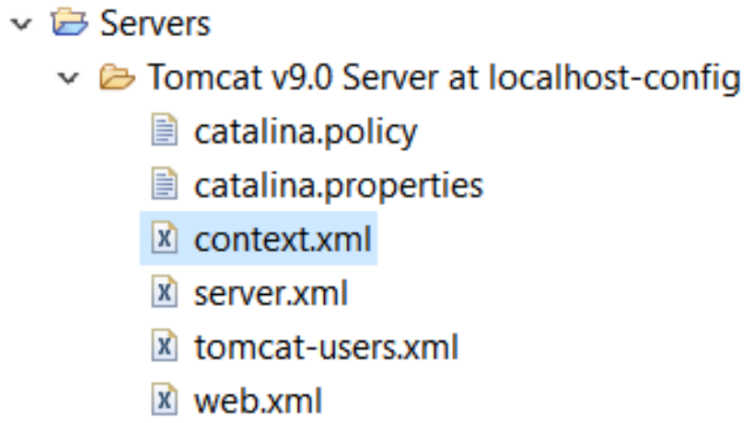

Imagen 4. Archivo context.xml
Fuente: Desafío Latam.

context.xml

Es el archivo de configuración propio del servidor *tomcat*. En él se configuran distintas funcionalidades del servidor, como por ejemplo, las credenciales de las bases de datos para generar un pool de conexiones. Dentro del archivo se definen las credenciales y datos de conexión a la base de datos tal como se muestra en la imagen 5.

```
<Resource name="jdbc/ConexionOracle" auth="Container" type="javax.sql.DataSource"
   maxActive="20" maxIdle="10" maxWait="5000"
   username="sys as sysdba" password="admin" driverClassName="oracle.jdbc.driver.OracleDriver"
   url="jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01"/>
```

Imagen 5. Definiendo las credenciales.
Fuente: Desafío Latam.

Con esas líneas lo que se está configurando es un *DataSource*. Los parámetros son:

1. **name**: Nombre del Datasource o fuente de datos.
2. **type**: tipo de Datasource en este caso Oracle.
3. **url**: Dirección del servidor y la base de datos.
4. **driverClassName**: Tipo de driver.
5. **username**: Usuario de conexión.
6. **password**: Clave de conexión.

Teniendo configurado el *DataSource* en el archivo context.xml del servidor, se puede apreciar que es lo que busca el método *generaPoolConexion* y la línea 39.

```
35    protected Connection generaPoolConexion() {
36        Context initContext;
37        try {
38            initContext = new InitialContext();
39            DataSource ds = (DataSource) initContext.lookup("java:/comp/env/jdbc/ConexionOracle");
40            try {
41                conn = ds.getConnection();
42            } catch (SQLException e) {
43                e.printStackTrace();
44            }
45        } catch (NamingException e) {
46            e.printStackTrace();
47        }
48        return conn;
49    }
```

Imagen 6. Obteniendo el DataSource.
Fuente: Desafío Latam.

Obteniendo el datasource se obtienen los datos de configuración, por eso la clase no define ningún parámetro de configuración.

A continuación la aplicación web debe tener conocimiento del pool de conexión y del datasource para poder trabajar con él. Para este tipo de configuraciones se utiliza un archivo de configuración de nombre *web.xml*, el cual se debe crear en la carpeta WEB-INF.

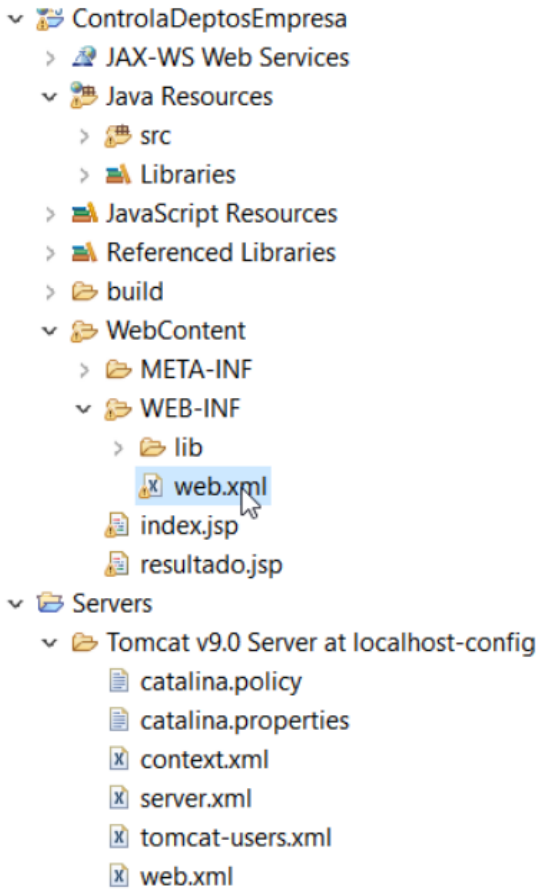

Imagen 7. Archivo web.xml.
Fuente: Desafío Latam.

Tal archivo *web.xml* debe tener las referencias del datasource mediante las siguientes líneas:

```
<resource-ref>
    <description>Pool conexiones</description>
    <res-ref-name>jdbc/ConexionOracle</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Imagen 8. Agregando las referencias al datasource.
Fuente: Desafío Latam.

La línea 3 contiene el nombre de *DataSource* configurado en el *context.xml*. Con estas configuraciones la aplicación ya está conectada mediante un pool de conexiones y un datasource.

Usar un pool de conexiones es más eficiente que abrir una nueva conexión cada vez que sea necesario, por lo tanto, no hay motivo para no usar un pool de conexiones. Ahora solo falta probar la aplicación. Recordar que con JDBC se utilizaba el método *obtenerConexion* desde la clase *DepartamentoDaoImp* en la línea 15.

Ahora para utilizar el *DataSource* e implementar el pool de conexiones, solamente llamar al método *generaPoolConexion()*.

```
12 public class DepartamentoDaoImp extends AdministradorConexion implements DepartamentoDao{
13
14     public DepartamentoDaoImp() {
15         Connection conn = generaPoolConexion();
16     }
17
```

Imagen 9. Utilizar el DataSource.
Fuente: Desafío Latam.