Geeks México

BLOG DE PROGRAMACIÓN EN ESPAÑOL SOBRE JAVA, FRAMEWORKS, BASES DE DATOS, CÓMPUTO EN LA NUBE, ETC. EN ESPAÑOL Y EN INGLÉS.

Spring boot + Spring JDBC

HACE 6 DÍAS DEJA UN COMENTARIO

En este post se explicará como acceder a una base de datos con Spring boot utilizando Spring JDBC paso a paso. Para esto se utilizará como base el proyecto Spring Boot + REST Jersey Parte 1 (https://geeks-mexico.com/2016/09/02/spring-boot-rest-jersey-parte-1/).

Paso 1 : Configurar las dependencias necesarias

El primer paso es agregar las dependencias necesarias para el proyecto, en este caso se necesitarán 2 **spring-boot-starter-jdbc** y **mysql-connector-java**.

```
<dependency>
    <groupId>org.springframework.boot</grou
    <artifactId>spring-boot-starter-jdbc</a
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artif
</dependency>
```

- ***spring-boot-starter-jdbc*** *: Contiene todas las clases necesarias para utilizar spring jdbc.*
- ***mysql-connector-java****: Es el driver de mysql para Java que permitirá conectarnos vía jdbc.*

Paso 2: Crear las tablas a utilizar en la base de datos

Para este ejemplo se utilizará como base de datos mysql para esto se creará una base de datos

llamada **jdbc_example** con la siguiente estructura:

```
CREATE TABLE USER(
USER_ID INTEGER PRIMARY KEY AUTO_INCREMENT,
USERNAME VARCHAR(100) NOT NULL,
PASSWORD VARCHAR(100) NOT NULL
);
```

Con los siguientes datos:

```
INSERT INTO USER (USERNAME,PASSWORD)VALUES(
INSERT INTO USER (USERNAME,PASSWORD)VALUES(
INSERT INTO USER (USERNAME,PASSWORD)VALUES(
```

Paso 3: Crear clase para representar el User

Ahora es necesario crear POJOS que representen los registros en la base de datos para esto se creará la clase **User.java**.

```
/**
 *
 */
package com.raidentrance.model;

/**
 * @author raidentrance
 *
 */
public class User {
    private Integer id;
    private String user;
    private String password;

    public User() {
    }

    public User(Integer id, String user, S
        super();
        this.id = id;
        this.user = user;
        this.password = password;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

```

```
33      public String getUser() {
34          return user;
35      }
36
37      public void setUser(String user) {
38          this.user = user;
39      }
40
41      public String getPassword() {
42          return password;
43      }
44
45      public void setPassword(String passwor
46          this.password = password;
47      }
48
49  }
```

Paso 4: Agregar la configuración de la base de datos al proyecto

El siguiente paso es incluir la configuración de la base de datos al proyecto, para esto es necesario editar el archivo application.properties con la siguiente configuración:

```
1  spring.datasource.url=jdbc:mysql://localhos
2  spring.datasource.username=root
3  spring.datasource.password=root
4  spring.datasource.driver-class-name=com.mys
```

Paso 5: Crear un DAO(Data access object)

Una vez que Spring conoce los datos de conexión, el siguiente paso es crear un DAO el cuál nos servirá para ejecutar todas las acciones sobre la base de datos.

```
1  /**
2   *
3   */
4  package com.raidentrance.dao;
5
6  import java.sql.Connection;
7  import java.sql.PreparedStatement;
8  import java.sql.ResultSet;
9  import java.sql.SQLException;
10 import java.util.List;
11
12 import javax.ws.rs.core.Response.Status;
```

```
13
14  import org.springframework.beans.factory.a
15  import org.springframework.dao.DataAccessE
16  import org.springframework.jdbc.core.JdbcT
17  import org.springframework.jdbc.core.Prepa
18  import org.springframework.jdbc.core.Resul
19  import org.springframework.jdbc.core.RowMa
20  import org.springframework.stereotype.Comp
21
22  import com.raidentrance.model.ServiceExcep
23  import com.raidentrance.model.User;
24
25  /**
26   * @author raidentrance
27   *
28   */
29  @Component
30  public class UserDao {
31
32      @Autowired
33      private JdbcTemplate jdbcTemplate;
34
35      public List<User> findAll() {
36          List<User> users = jdbcTemplate.qu
37              @Override
38              public User mapRow(ResultSet r
39                  User user = new User(rs.ge
40                  return user;
41              }
42          });
43          return users;
44      }
45
46      public User findByUsername(String user
47          User user = jdbcTemplate.query(new
48
49              @Override
50              public PreparedStatement creat
51                  PreparedStatement ps = con
52                  ps.setString(1, username);
53                  return ps;
54              }
55          }, new ResultSetExtractor<User>()
56              @Override
57              public User extractData(Result
58                  if (rs.next()) {
59                      User user = new User(r
60                      return user;
61                  } else {
62                      return null;
63                  }
64              }
65          });
66          if (user != null) {
67              return user;
68          } else {
69              throw new ServiceException(Sta
70          }
71      }
72  }
```

En el código anterior se pueden observar los siguientes puntos importantes:

- *La anotación* ***@Component*** *: Significa que tendremos un objeto de esa clase viviendo dentro del contexto de Spring y que no será necesario utilizar el operador new para crearlo.*
- ***@Autowired*** *JdbcTemplate jdbcTemplate : Indica que utilizando las configuraciones que se definieron en el archivo application.properties se creará un template de la clase* ***JdbcTemplate(****El cual es parte de spring data****)*** *y se inyectará en la referencia* ***jdbcTemplate*** *para que lo utilicemos.*
- *public List* ***findAll()*** *: Este método será utilizado para buscar todos los usuarios que se encuentren en la tabla. Como se puede observar este objeto recibe el query que se desea ejecutar y un objeto del tipo* ***RowMapper*** *el cual define como se va a traducir de un ResultSet a un objeto Java de tipo* ***User****.*
- *public User* ***findByUsername****(String username): Del mismo modo el método findByUsername será utilizado para buscar en la base de datos al usuario que tiene el username especificado. En este ejemplo se puede apreciar que se utiliza un* ***PreparedStatement*** *para prevenir SQL injection debido a que esta consulta recibe parámetros. Otro punto diferente es que a diferencia del método findAll() este utiliza un ResultSetExtractor en lugar de un RowMapper debido a que solo se espera un resultado en la repuesta.*

- *Por último podemos ver que en caso de que no se encuentre el usuario se arrojará una excepción de tipo ServiceException con el mensaje, código y estatus http.*

Paso 6: Utilizar el DAO en nuestro servicio

Una vez que ya se creo el DAO el siguiente paso es utilizarlo en nuestro endpoint, en futuros posts se verá que es mejor separarlo en servicios pero por ahora se inyectará en el endpoint UserResource.

```
1  /**
2   *
3   */
4  package com.raidentrance.resource;
5
6  import javax.ws.rs.Consumes;
7  import javax.ws.rs.GET;
8  import javax.ws.rs.Path;
9  import javax.ws.rs.PathParam;
10 import javax.ws.rs.Produces;
11 import javax.ws.rs.core.MediaType;
12 import javax.ws.rs.core.Response;
13
14 import org.slf4j.Logger;
15 import org.slf4j.LoggerFactory;
16 import org.springframework.beans.factory.a
17 import org.springframework.stereotype.Comp
18
19 import com.raidentrance.dao.UserDao;
20 import com.raidentrance.model.ServiceExcep
21
22 /**
23  * @author raidentrance
24  *
25  */
26
27 @Compo
28 @Path( /users )
29 @Consumes(MediaType.APPLICATION_JSON)
30 @Produces(MediaType.APPLICATION_JSON)
31 public class UserResource {
32
33     @Autowired
34     private UserDao userDao;
35
36     private static final Logger log = Logg
37
38     @GET
```

```
39      public Response getUsers() {
40          log.info("Getting users");
41          return Response.ok(userDao.findAl
42      }
43
44      @GET
45      @Path("/user/{username}")
46      public Response getUser(@PathParam("us
47          log.info("Getting users");
48          return Response.ok(userDao.findByl
49      }
50
51  }
```

Como se puede observar para inyectar el objeto de tipo UserDao lo único que se debe hacer es utilizar la anotación ***@Autowired*** *ya que el objeto ya vive dentro del contexto de Spring.*

Paso 7: Probando todo junto

Para ejecutar la aplicación se debe ejecutar la clase principal del mismo modo que en cualquier aplicación spring boot y se accederá a la siguiente url http://localhost:8080/users (http://localhost:8080/users) la cual mostrará lo siguiente:


```
[
  - {
        id: 1,
        user: "raidentrance",
        password: "superSecret"
    },
  - {
        id: 2,
        user: "john",
        password: "smith"
    },
  - {
        id: 3,
        user: "juan",
        password: "hola123"
    }
]
```


Y para obtener solo un usuario se utilizará la url http://localhost:8080/users/user/raidentrance

(http://localhost:8080/users/user/raidentrance) con la siguiente salida:

```
{
    id: 1,
    user: "raidentrance",
    password: "superSecret"
}
```

El código completo lo puedes encontrar en https://github.com/raidentrance/spring-boot-example/tree/part6-spring-jdbc (https://github.com/raidentrance/spring-boot-example/tree/part6-spring-jdbc).

Si quieres aprender más sobre web services o Spring boot recomendamos los siguientes libros:

- Java Web Services: Up and Running
- Pro Spring Boot

Autor: Alejandro Agapito Bautista

Twitter: @raidentrance (https://geeksjavamexico.wordpress.com/mentions/raidentrance/)

Contacto:raidentrance@gmail.com