链滴

Spring 与 JDBC

作者：jiangyue

原文链接：https://ld246.com/article/1479817901136

来源网站：链滴

将数据访问的功能放到一个或多个专注于此项任务的组件，这样的组件称数据访问对象。

编写良好的数据访问对象应该以接口的方式暴露功能，Spring 遵循面向对中的 针对接口编程，将持久层隐藏在接口之后。

Spring 提供了统一的异常体系，提供了多个数据访问异常，分别描述异常生所对应的问题，可以用在所支持的所有持久化方案中。

Spring 将数据访问过程中固定的和可变的部分划分模板(template)和回调(callback)，将过程中与特定实现部分委托给接口，用接口的不同实现定义过程中的具体行为。

Spring 提供了在上下文中配置数据源 bean 的多种方式，包括过 JDBC 驱动程序定义的数据源，通过 JNDI 查找的数据源 和 <strog>连接池的数据源。

位于 jee 命名空间的 <code><jee:jndi-lookup></code> 元素可用于检索 JNDI 中的对象(包括数据源)并装配到 Spring，如 <code><jee:jndi-lookup id="dataSource" jndi-name="/jdbc/xxxDS" resource-ref="tru"></code> 。resource-ref 属性为 true 时，jndi-name 将会自动添加"java:comp/en/"前缀。

JavaConfig 配置 JNDI：
<pre><code> @Bean
  public JndiObjectFactoryBean dataSource() {
      JndiObjectFactoryBean jndiObjectFB = new JndiObjectFactoryBean();
      jndiObjectFB.setJndiname("jdbc/xxxDS");
      jndiObjectFB.setResourceRef(true);
      jndiObjectFB.setProxyInterface(javax.sql.DataSource.class);
      return jndiObjectFB;
  }
</code></pre>

Spring 中可直接配置数据源连接池，虽然 Spring 中未提供数据源连接池的实现，但有多项可用案，如c3p0、BoneCP 等开源实现。

Spring 提供了通过 JDBC 驱动定义数据源的类，例如:
<pre><code> @Bean
  public DataSource dateSource() {
      DriverManagerDataSource ds = new DriverManagerDataSource();
      ds.setDriverClassName("org.h2.Driver");
      ds.setUrl("jdbc:xxxxx");

```
    ds.setUsername("root");
    ds.setPassword("");
    return ds;
 }
</code></pre>
</li>
<li>
<p>配合 <code>@Profile</code> 注解可以从多个数据源中选择需要的数据源。</p>
</li>
<li>
<p>Spring 的 JDBC 框架负责<strong>管理资源</strong>和<strong>处理异常</strong>，开者只需编写<strong>从数据库读写数据</strong>的必须代码，从而简化 JDBC 代码。需要使用命参数时，需要使用 <code>NamedParameterJdbcTemplate</code>，对于多数 JDBC 任务来说 <ode>JdbcTemplate</code> 就是最好的选择。</p>
</li>
<li>
<p>JdbcTemplate 需要 <strong>DataSource</strong> 参数，可以在需要 JDBCTemplate 的位<strong>声明 JdbcOperations 接口</strong>，并<strong>注入 JdbcTemplate 对象</strong
。</p>
</li>
<li>
<p><strong>RowMapper</strong> 用于从 ResultSet 中提<strong>取数据并构建域对象</stro
g>，如</p>
<pre><code>  private static final class SpitterRowMapper implements RowMapper&amp;lt;Spitte
&amp;gt; {
    public Spitter mapRow(ResultSet rs, int rowNum) throws SQLException {
      long id = rs.getLong("id");
      String username = rs.getString("username");
      String password = rs.getString("password");
      String fullName = rs.getString("fullname");
      String email = rs.getString("email");
      boolean updateByEmail = rs.getBoolean("updateByEmail");
      return new Spitter(id, username, password, fullName, email, updateByEmail);
    }
 }
</code></pre>
</li>
<li>
<p>几个 JdbcTemplate 使用示例：</p>
<pre><code>  public long count() {
    return jdbcTemplate.queryForLong("select count(id) from Spitter");
 }
<p>public Spitter save(Spitter spitter) {<br>
Long id = spitter.getId();<br>
if (id == null) {<br>
long spitterId = insertSpitterAndReturnId(spitter);<br>
return new Spitter(spitterId, spitter.getUsername());<br>
} else {<br>
jdbcTemplate.update("update Spitter set username=?",<br>
spitter.getUsername(),<br>
spitter.getPassword(),<br>
id);<br>
}<br>
return spitter;<br>
```

}
private long insertSpitterAndReturnId(Spitter spitter) {
SimpleJdbcInsert jdbcInsert = new SimpleJdbcInsert(jdbcTemplate).withTableName("Spitter")

jdbcInsert.setGeneratedKeyName("id");
Map<String, Object> args = new HashMap<String, Object>();
args.put("username", spitter.getUsername());
args.put("password", spitter.getPassword());
long spitterId = jdbcInsert.executeAndReturnKey(args).longValue();
return spitterId;
}
public Spitter findOne(long id) {
return jdbcTemplate.queryForObject(
SELECT_SPITTER + " where id=?", new SpitterRowMapper(), id);
}
</code> <code>public void delete(long id) {
jdbcTemplate.update("delete from Spittle where id=?", id);
}
</code> </pre>

