**《Spring JDBC详解》**

本文旨在讲述Spring JDBC模块的用法。Spring JDBC模块是Spring框架的基础模块之一。

作者：chszs，转载需注明。博客主页：<http://blog.csdn.net/chszs>

**一、概述**

在Spring JDBC模块中，所有的类可以被分到四个单独的包：  
1）core  
即核心包，它包含了JDBC的核心功能。此包内有很多重要的类，包括：JdbcTemplate类、SimpleJdbcInsert类，SimpleJdbcCall类，以及NamedParameterJdbcTemplate类。  
2）datasource  
即数据源包，访问数据源的实用工具类。它有多种数据源的实现，可以在JavaEE容器外部测试JDBC代码。  
3）object  
即对象包，以面向对象的方式访问数据库。它允许执行查询并返回结果作为业务对象。它可以在数据表的列和业务对象的属性之间映射查询结果。  
4）support  
即支持包，是core包和object包的支持类。例如提供了异常转换功能的SQLException类。  
  
**二、配置**  
下面我们以MySQL数据库为例，开始简单的数据源配置：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. @Configuration
2. @ComponentScan("com.ch.myalbumjdbc")
3. **public** **class** SpringJdbcConfig {
4. @Bean
5. **public** DataSource mysqlDataSource() {
6. DriverManagerDataSource dataSource = **new** DriverManagerDataSource();
7. dataSource.setDriverClassName("com.mysql.jdbc.Driver");
8. dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc");
9. dataSource.setUsername("guest\_user");
10. dataSource.setPassword("guest\_password");

13. **return** dataSource;
14. }
15. }

或者，您还可以利用嵌入式数据库进行开发或测试，比如用HSQL嵌入式数据库进行快速配置并创建实例：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. @Bean
2. **public** DataSource dataSource() {
3. **return** **new** EmbeddedDatabaseBuilder()
4. .setType(EmbeddedDatabaseType.HSQL)
5. .addScript("classpath:jdbc/schema.sql")
6. .addScript("classpath:jdbc/test-data.sql").build();
7. }

最后，也可以使用XML配置来实现前面的注释配置的效果：

**[html]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **<bean** id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
2. destroy-method="close"**>**
3. **<property** name="driverClassName" value="com.mysql.jdbc.Driver"**/>**
4. **<property** name="url" value="jdbc:mysql://localhost:3306/springjdbc"**/>**
5. **<property** name="username" value="guest\_user"**/>**
6. **<property** name="password" value="guest\_password"**/>**
7. **</bean>**

**三、JdbcTemplate的使用和运行查询**  
1、基本的查询  
JDBC模板是Spring JDBC模块中主要的API，它提供了常见的数据库访问功能：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **int** result = jdbcTemplate.queryForObject(
2. "SELECT COUNT(\*) FROM EMPLOYEE", Integer.**class**);

下面是简单的插入功能：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **public** **int** addEmplyee(**int** id) {
2. **return** jdbcTemplate.update(
3. "INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)", 5, "Bill", "Gates", "USA");
4. }

注意提供参数的标准语法——使用“？”字符。下面，让我们看看替代语法。  
  
2、查询与命名参数  
要获得命名参数的支持，我们需要使用Spring JDBC提供的其它JDBC模板——NamedParameterJdbcTemplate。  
此类封装了JbdcTemplate，并提供了使用“？”来替代指定参数的传统语法。它使用传递的参数来替换占位符“？”，以执行传参的查询：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. SqlParameterSource namedParameters = **new** MapSqlParameterSource().addValue("id", 1);
2. **return** namedParameterJdbcTemplate.queryForObject(
3. "SELECT FIRST\_NAME FROM EMPLOYEE WHERE ID = :id", namedParameters, String.**class**);

请注意，我们使用的是MapSqlParameterSource来提供值的命名参数。  
下面是使用bean类的属性来确定命名参数简单例子：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. Employee employee = **new** Employee();
2. employee.setFirstName("James");
4. String SELECT\_BY\_ID = "SELECT COUNT(\*) FROM EMPLOYEE WHERE FIRST\_NAME = :firstName";
6. SqlParameterSource namedParameters = **new** BeanPropertySqlParameterSource(employee);
7. **return** namedParameterJdbcTemplate.queryForObject(SELECT\_BY\_ID, namedParameters, Integer.**class**);

请注意，我们是怎样利用BeanPropertySqlParameterSource的实现来替代指定的命名参数。  
  
3、把查询结果映射到Java对象  
还有一个非常有用的功能是把查询结果映射到Java对象——通过实现RowMapper接口。  
例如，对于查询返回的每一行结果，Spring会使用该行映射来填充Java bean：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **public** **class** EmployeeRowMapper **implements** RowMapper<Employee> {
2. @Override
3. **public** Employee mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {
4. Employee employee = **new** Employee();

7. employee.setId(rs.getInt("ID"));
8. employee.setFirstName(rs.getString("FIRST\_NAME"));
9. employee.setLastName(rs.getString("LAST\_NAME"));
10. employee.setAddress(rs.getString("ADDRESS"));

13. **return** employee;
14. }
15. }

现在，我们传递行映射器给查询API，并得到完全填充好的Java对象：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. String query = "SELECT \* FROM EMPLOYEE WHERE ID = ?";
2. List<Employee> employees = jdbcTemplate.queryForObject(
3. query, **new** Object[] { id }, **new** EmployeeRowMapper());

**四、异常转换**  
Spring提供了自己的开箱即用的数据异常分层——DataAccessException作为根异常，它负责转换所有的原始异常。  
所以开发者无需处理底层的持久化异常，因为Spring JDBC模块已经在DataAccessException类及其子类中封装了底层的异常。  
这样可以使异常处理机制独立于当前使用的具体数据库。  
除了默认的SQLErrorCodeSQLExceptionTranslator类，开发者也可以提供自己的SQLExceptionTranslator实现。  
下面是一个自定义SQLExceptionTranslator实现的简单例子，当出现完整性约束错误时自定义错误消息：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **public** **class** CustomSQLErrorCodeTranslator **extends** SQLErrorCodeSQLExceptionTranslator {
2. @Override
3. **protected** DataAccessException customTranslate
4. (String task, String sql, SQLException sqlException) {
5. **if** (sqlException.getErrorCode() == -104) {
6. **return** **new** DuplicateKeyException(
7. "Custom Exception translator - Integrity constraint violation.", sqlException);
8. }
9. **return** **null**;
10. }
11. }

要使用自定义的异常转换器，我们需要把它传递给JdbcTemplate——通过callingsetExceptionTranslator()方法：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. CustomSQLErrorCodeTranslator customSQLErrorCodeTranslator = **new** CustomSQLErrorCodeTranslator();
2. jdbcTemplate.setExceptionTranslator(customSQLErrorCodeTranslator);

**五、使用SimpleJdbc类实现JDBC操作**  
SimpleJdbc类提供简单的方法来配置和执行SQL语句。这些类使用数据库的元数据来构建基本的查询。 SimpleJdbcInsert类和SimpleJdbcCall类提供了更简单的方式来执行插入和存储过程的调用。  
1、SimpleJdbcInsert类  
下面，让我们来看看执行简单的插入语句的最低配置，基于SimpleJdbcInsert类的配置产生的INSERT语句。  
所有您需要提供的是：表名、列名和值。让我们先创建SimpleJdbcInsert：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. SimpleJdbcInsert simpleJdbcInsert = **new** SimpleJdbcInsert(dataSource).withTableName("EMPLOYEE");

现在，让我们提供列名和值，并执行操作：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **public** **int** addEmplyee(Employee emp) {
2. Map<String, Object> parameters = **new** HashMap<String, Object>();
3. parameters.put("ID", emp.getId());
4. parameters.put("FIRST\_NAME", emp.getFirstName());
5. parameters.put("LAST\_NAME", emp.getLastName());
6. parameters.put("ADDRESS", emp.getAddress());

9. **return** simpleJdbcInsert.execute(parameters);
10. }

为了让数据库生成主键，我们可以使用executeAndReturnKey() API，我们还需要配置的实际自动生成的列：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. SimpleJdbcInsert simpleJdbcInsert = **new** SimpleJdbcInsert(dataSource)
2. .withTableName("EMPLOYEE")
3. .usingGeneratedKeyColumns("ID");

6. Number id = simpleJdbcInsert.executeAndReturnKey(parameters);
7. System.out.println("Generated id - " + id.longValue());

最后，我们还能使用BeanPropertySqlParameterSource和MapSqlParameterSource传递数据。  
  
2、用SimpleJdbcCall调用存储过程  
让我们看看如何执行存储过程——我们使用SimpleJdbcCall的抽象：

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. SimpleJdbcCall simpleJdbcCall = **new** SimpleJdbcCall(dataSource)
2. .withProcedureName("READ\_EMPLOYEE");

5. **public** Employee getEmployeeUsingSimpleJdbcCall(**int** id) {
6. SqlParameterSource in = **new** MapSqlParameterSource().addValue("in\_id", id);
7. Map<String, Object> out = simpleJdbcCall.execute(in);

10. Employee emp = **new** Employee();
11. emp.setFirstName((String) out.get("FIRST\_NAME"));
12. emp.setLastName((String) out.get("LAST\_NAME"));

15. **return** emp;
16. }

**六、批处理操作**  
另一个简单的用例——把多种操作合在一起实现批处理  
1、使用JdbcTemplate执行基本的批处理操作  
使用JdbcTemplate类，通过batchUpdate() API来执行基本的批处理操作：  
注意BatchPreparedStatementSetter实现是很有趣的。

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. **public** **int**[] batchUpdateUsingJdbcTemplate(List<Employee> employees) {
2. **return** jdbcTemplate.batchUpdate("INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)",
3. **new** BatchPreparedStatementSetter() {
4. @Override
5. **public** **void** setValues(PreparedStatement ps, **int** i) **throws** SQLException {
6. ps.setInt(1, employees.get(i).getId());
7. ps.setString(2, employees.get(i).getFirstName());
8. ps.setString(3, employees.get(i).getLastName());
9. ps.setString(4, employees.get(i).getAddress();
10. }
11. @Override
12. **public** **int** getBatchSize() {
13. **return** 50;
14. }
15. });
16. }

2、使用NamedParameterJdbcTemplate执行批处理操作  
对于批处理操作，还可以选择使用NamedParameterJdbcTemplate的batchUpdate() API来执行。  
此API比先前的更简单——无需实现任何额外的接口来设置参数，因为它有一个内部的预准备语句的setter来传递预设的参数值。  
参数值可以通过batchUpdate()方法传递给SqlParameterSource的数组。

**[java]** [view plain](http://blog.csdn.net/chszs/article/details/43971981) [copy](http://blog.csdn.net/chszs/article/details/43971981)

[print](http://blog.csdn.net/chszs/article/details/43971981)[?](http://blog.csdn.net/chszs/article/details/43971981)[在CODE上查看代码片](https://code.csdn.net/snippets/608849)

1. SqlParameterSource[] batch = SqlParameterSourceUtils.createBatch(employees.toArray());
2. **int**[] updateCounts = namedParameterJdbcTemplate.batchUpdate(
3. "INSERT INTO EMPLOYEE VALUES (:id, :firstName, :lastName, :address)", batch);
4. **return** updateCounts;

**七、结论**  
本文讲述了Spring框架中的JDBC抽象，覆盖了Spring JDBC模块内建的各种功能和实际的例子。