JdbcTemplate 类的实例是*线程安全*配置的。所以可以配置 JdbcTemplate 的单个实例，然后将这个共享的引用安全地注入到多个 DAOs 中。常见的做法是在你的 Spring 配置文件中配置数据源，然后共享数据源 bean 依赖注入到 DAO 类中，并在数据源的设值函数中创建了 JdbcTemplate。

XML 文件中配置数据源：

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
   <property name="url" value="jdbc:mysql://localhost:3306/TEST"/>
   <property name="username" value="root"/>
   <property name="password" value="password"/>
</bean>
```

数据访问对象（DAO）

DAO 代表常用的数据库交互的数据访问对象。

执行 SQL 语句（CRUD）

查询一个整数类型：

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL );
```

查询一个 long 类型：

```
String SQL = "select count(*) from Student";
long rowCount = jdbcTemplateObject.queryForLong( SQL );
```

一个使用绑定变量的简单查询：

```
String SQL = "select age from Student where id = ?";
int age = jdbcTemplateObject.queryForInt(SQL, new Object[]{10});
```

查询字符串：

```
String SQL = "select name from Student where id = ?";
String name = jdbcTemplateObject.queryForObject(SQL, new Object[]{10},
String.class);
```

查询并返回一个对象：

```
String SQL = "select * from Student where id = ?";
Student student = jdbcTemplateObject.queryForObject(SQL,
                  new Object[]{10}, new StudentMapper());
public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setID(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));
      return student;
   }
}
```

查询并返回多个对象：

```
String SQL = "select * from Student";
List<Student> students = jdbcTemplateObject.query(SQL,
                         new StudentMapper());
public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setID(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));
      return student;
   }
}
```

在表中插入一行：

```
String SQL = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );
```

更新表中的一行：

```
String SQL = "update Student set name = ? where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );
```

从表中删除一行：

```
String SQL = "delete Student where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{20} );
```

执行 DDL 语句

使用 *jdbcTemplate* 中的 execute(..) 方法来执行任何 SQL 语句或 DDL 语句。下面是一个使用 CREATE 语句创建一个表的示例：

```
String SQL = "CREATE TABLE Student( " +
   "ID   INT NOT NULL AUTO_INCREMENT, " +
   "NAME VARCHAR(20) NOT NULL, " +
   "AGE  INT NOT NULL, " +
   "PRIMARY KEY (ID));"
jdbcTemplateObject.execute( SQL );
```