吴老师教学讲义

忽然抚尺一下，群响毕绝。撤屏视之，一人、一桌、一椅、一扇、一抚尺而已

吴　青

QQ:16910735

wuqing_bean@126.com

http://blog.sina.com/accpwulaoshi

Spring 中的 JDBC/DAO 组件

在 Spring 框架的 7 个模块组件中其中有一个 Spring JDBC/DAO 组件，它提供了对 JDBC 的支持,它使用数据访问对象（DAO）模式实现数据层的访问。它的目标在采用统一的方式来使用不同的数据访问技术， 如 JDBC，Hibernate 或者 JDO 等。它不仅可以让我们方便地在这些持久化技术间切换， 而且在编码的时候不用考虑处理各种技术中特定的异常。

1. Spring JDBC 包结构

Spring Framework 的 JDBC 抽象框架由四个包构成：

- org.springframework.jdbc.core

Spring 为了统一数据访问，在设计的时候，采用了模板模式，将常用的数据访问都做成了模板，我们在编码的时候，使用的模板类都包含在这个包以及它的子包中。

- org.springframework.jdbc.datasource

供了一些工具类来简化对 DataSource（数据源）的访问。同时提供了多种简单的 DataSource 实现。

- org.springframework.jdbc.object

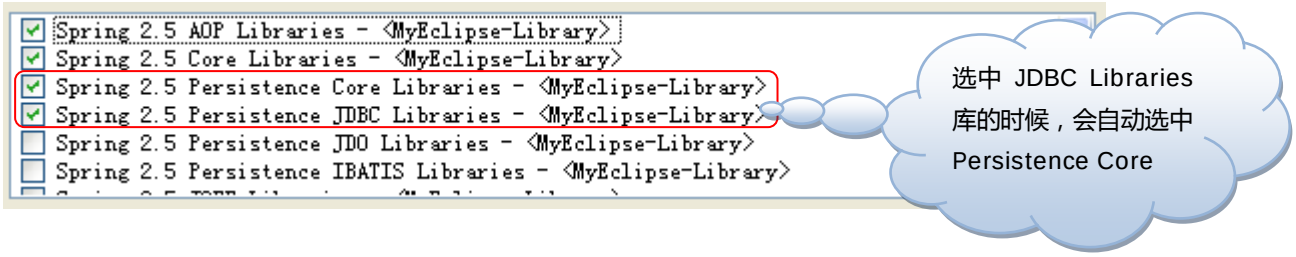
包含了一些类，用于将 RDBMS 查询、更新以及存储过程表述为一些可重用的、线程安全的对象。

- org.springframework.jdbc.support

定义了 SQLException 转化类以及一些其他的工具类。在 JDBC 调用过程中所抛出的异常都会被转化为在 org.springframework.dao 包中定义的异常。也就是说，凡是使用 Spring 的 JDBC 封装层的代码无需实现任何 JDBC 或者 RDBMS 相关的异常处理。所有的这些被转化的异常都是 unchecked 异常

这些包都存放在 SPRING_HOME/dist/modules/spring-jdbc.jar 文件中。所以在使用 Spring JDBC 的时候，需要加入这个 jar 文件到 Classpath 中。

新建 java 项目 ,加入 Spring 支持。在加入 Spring 支持的时候注意选择:Spring JDBC Libraries。这样 MyEclipse 就会往项目中加入我们所需的 jar 文件，同时也加入了 DBCP 连接池和 c3p0 连接池的 jar 文件。



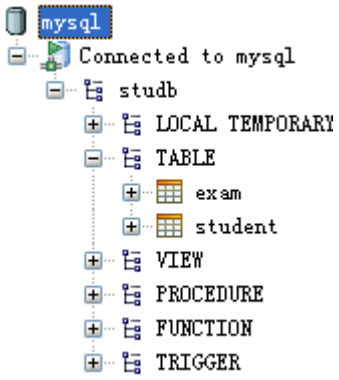
2. 数据源 DataSource 接口

为了从数据库中取得数据，我们首先需要获取一个数据库连接。Spring 通过 DataSource 对象来完成这个工作。 DataSource 是 JDBC 规范的一部分，它被视为一个通用的数据库连接工厂。

DataSource 有多种实现类，Spring 提供了 DriverManagerDataSource 实现类，它可以方便我们开发的时候进行测试，也可以使用 DBCP 连接池提供的 DataSource 实现类 org.apache.commons.dbcp.BasicDataSource，c3p0 连接池也提供了 DataSource 的实现类 com.mchange.v2.c3p0.ComboPooledDataSource。

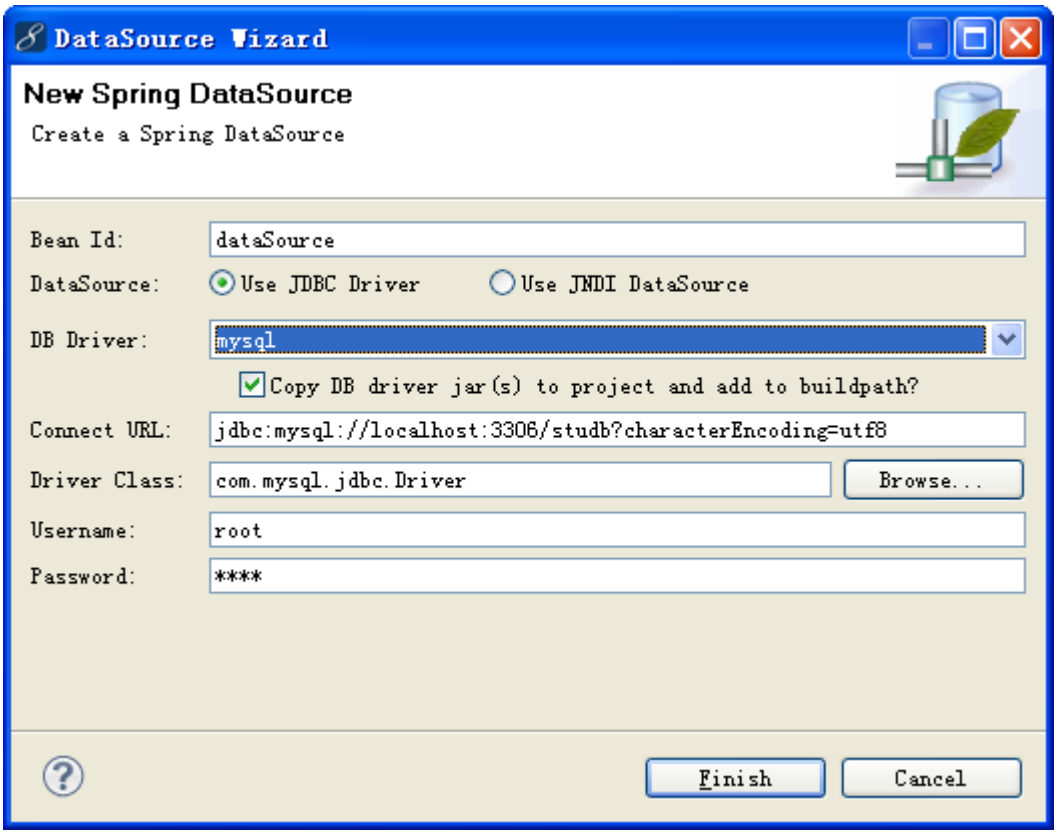
所以只需要在 Spring 配置文件中将 DataSource 创建出来即可。

2.1 在 MyEclipse 的 Database Explorer 中配置到数据库的连接



2.2 Spring 配置文件中添加数据源

在 Spring 配置文件中单击鼠标右键



下面是生成的代码:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
      <property name="driverClassName" value="com.mysql.jdbc.Driver">
      </property>
      <property name="url"    value="jdbc:mysql://localhost:3306/studb?characterEncoding=utf8">
      </property>
      <property name="username" value="root"></property>
      <property name="password" value="root"></property>
</bean>
```

MyEclipse默认使用的是DBCP连接池 ,我们也可以使用Spring自己的DataSource实现，只需要将类名换成：org.springframework.jdbc.datasource.DriverManagerDataSource即可。

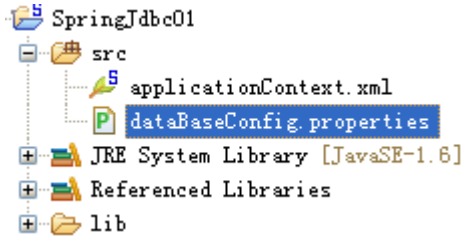
上面的 DBCP 配置只是一部分，下面给出完配置了连接数的配置：

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
<!-- JDBC驱动程序类名 -->
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <!-- 连接字符串 -->
    <property name="url"
        value="jdbc:mysql://localhost:3306/studb?characterEncoding=utf8">
    </property>
    <!-- 数据源用户名 -->
    <property name="username" value="root"></property>
    <!-- 数据源密码 -->
    <property name="password" value="root"></property>
    <!-- 初始连接数量 -->
    <property name="initialSize" value="50"></property>
    <!-- 最大连接数量 -->
    <property name="maxActive" value="80"></property>
</bean>
```

2.3 从 properties 配置文件中读取信息

数据库的配置在开发和实施过程中，经常发生改变，我们可以将数据库配置信息单独配置在外部的 properties 文件中，使用时直接利用 properties 中的键引用其值即可。Spring 提供了 PropertyPlaceholderConfigurer，该 Bean 注册到容器之后，在容器启动时加载配置的 properties 文件。它是 BeanFactoryPostProcessor 的一种，是 Spring 容器的扩展。

在 src 中新建一个 dataBaseConifg.properties 文件：




```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/studb?characterEncoding=utf8
jdbc.user=root
jdbc.password=root
```

在配置文件中的其他位置可以使用“${资源文件中的键}”表达式，获取对应的资源文件中的值。下面是引用了资源文件配置信息的 C3P0 数据源的配置。

在 Spring 配置文件 applicationContext.xml 文件中配置 BeanFactoryPostProcessor 读取属性文件。

```
<bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>/dataBaseConfig.properties</value>
        </list>
    </property>
</bean>
```

下面再给出 c3p0 连接池的配置

```
<bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass">
        <value>${jdbc.driver}</value>
    </property>
    <property name="jdbcUrl">
        <value>${jdbc.url}</value>
    </property>
    <property name="user">
        <value>${jdbc.user}</value>
    </property>
    <property name="password">
        <value>${jdbc.password}</value>
    </property>
    <property name="maxPoolSize" value="30"></property>
    <property name="initialPoolSize" value="5"></property>
</bean>
```

3. JdbcTemplate 类

JdbcTemplate 是 core 包的核心类。它替我们完成了资源的创建以及释放工作，从而简化了我们对 JDBC 的使用。 它还可以帮助我们避免一些常见的错误，比如忘记关闭数据库连接。 JdbcTemplate 将完成 JDBC 核心处理流程，比如创建数据库连接，Statement 对象的创建，SQL 语句的执行，而把 SQL 语句的编写及查询结果的提取工作留给我们的应用代码。 它可以完成 SQL 查询、更新以及调用存储过程，可以对结果集 ResultSet 进行遍历并加以提取。 它还可以捕获 JDBC 异常并将其转换成 org.springframework.dao 包中定义的，通用的，信息更丰富的异常。

实际编码的时候，要使用 Jdbctemplate 对象来完成 jdbc 操作。通常情况下，有三种种方式得到 JdbcTemplate 对象。

第一种方式：我们可以在自己定义的 DAO 实现类中注入一个 DataSource 引用来完成 JdbcTemplate 的实例化。也就是它是从外部“注入”DataSource 到 DAO 中，然后自己实例化 JdbcTemplate，然后将 DataSource 设置到 JdbcTemplate 对象中。

第二种方式： 在 Spring 的 IoC 容器中配置一个 JdbcTemplate 的 bean，将 DataSource 注入进来，然后再把 JdbcTemplate 注入到自定义 DAO 中。

第三种方式: Spring 提供了 org.springframework.jdbc.core.support.JdbcDaoSupport 类，这个类中定义了 JdbcTemplate 属性，也定义了 DataSource 属性，当设置 DataSource 属性的时候，会创建 jdbcTemplate 的实例，所以我们自己编写的 DAO 只需要继承 JdbcDaoSupport 类，然后注入 DataSource 即可。

我们可以采用第三种方式对数据库进行操作。

最后注意 JdbcTemplate 中使用的所有 SQL 将会以“DEBUG”级别记入日志。

3.1 常用方法

- ☑ int[] batchUpdate(String[] sql)：批量执行 insert、update、delete 语句。
- ☑ void execute(String sql)：执行一个独立的 sql 语句，包括 DDL 语句。
- ☑ List<T> query(String sql,Object[] args,RowMapper mapper)：执行一个 select 语句，并且实现 RowMapper 接口用于对象关系映射，返回一个具体对象的列表。其中可以不带参数 args。
- ☑ int queryForInt(String sql,Object[] args)：执行返回一个整数的查询语句。Spring 提供了一系列 queryForXXX(...)的方法，可以执行返回某个对象或其他数据的操作。
- ☑ int update(String sql,Object[] args)：执行一个 insert，update 或者 delete 语句，返回该语句影响的数据库行数。

JdbcTemplate 类的实例是线程安全的实例,更多的方法请参考 Spring API 文档

4. 使用 JdbcTemplate 类

下面演示如何通过 JdbcTemplate 类完成数据库的访问,为了能看到执行过程中的 sql 语句，我们加入 Log4j 的配置文件，然后设置级别为 Debug 级别。

4.1 准备一个实体类:

```
public class Student {
    private int stuID;
    private String stuName;
    private int stuAge;
    //省略getter/setter方法
}
```

4.2 编写 DAO

编写的 DAO 继承自 Spring 提供的 JdbcDaoSupport 类

```
public class UserDAO extends JdbcDaoSupport {
    public void saveStudent(Student student){
        String sql="insert into student(stuName,stuAge) values(?,?)";
        this.getJdbcTemplate().update(sql,new Object[]{student.getStuName(),student.getStuAge()} );
    }
    public List<Student> findAll(){
        String sql="select * from student";
        //第二个参数需要一个行转换器，行转换器要实现RowMapper接口
        //这里采用匿名内部类的形式
        List<Student> list=this.getJdbcTemplate().query(sql, new RowMapper() {
            @Override
            public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
                Student stu=new Student();
                stu.setStuID(rs.getInt("stuID"));
                stu.setStuName(rs.getString("stuName"));
                stu.setStuAge(rs.getString("stuAge"));
                return stu;
            }
        });
        return list;
    }
    public int getStudentCount(){
        String sql="select count(*) from student";
        int count=this.getJdbcTemplate().queryForInt(sql);
        return count;
    }
}
```

4.3 配置文件中配置

让 Spring 为自己的 DAO 注入 DataSource

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <!-- JDBC驱动程序类名 -->
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <!-- 连接字符串 -->
    <property name="url"
        value="jdbc:mysql://localhost:3306/studb?characterEncoding=utf8">
    </property>
    <!-- 数据源用户名 -->
    <property name="username" value="root"></property>
    <!-- 数据源密码 -->
    <property name="password" value="root"></property>
    <!-- 初始连接数量 -->
    <property name="initialSize" value="50"></property>
    <!-- 最大连接数量 -->
    <property name="maxActive" value="80"></property>
</bean>
<bean id="studentDAO" class="com.wq.dao.StudentDAO">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

4.4 启动运行

```
public static void main(String[] args) {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "/applicationContext.xml");
    StudentDAO sdao = (StudentDAO) ctx.getBean("studentDAO");
    Student stu = new Student("张旺财", 20);
    sdao.saveStudent(stu);
    for (Student student : sdao.findAll()) {
        System.out.println(student.getStuName());
    }
    System.out.println("总记录数:" + sdao.getStudentCount());
}
```