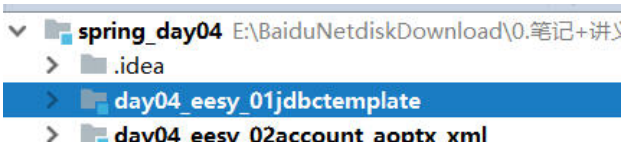
第1章 Spring 中的 JdbcTemplate[会用]

1.1 JdbcTemplate 概述



它是 spring 框架中提供的一个对象，是对原始 Jdbc API 对象的简单封装。spring 框架为我们提供了很多的操作模板类。

操作关系型数据的：

JdbcTemplate

HibernateTemplate

操作 nosql 数据库的：

RedisTemplate

操作消息队列的：

JmsTemplate

我们今天的主角在 spring-jdbc-5.0.2.RELEASE.jar 中，我们在导包的时候，除了要导入这个 jar 包外，还需要导入一个 spring-tx-5.0.2.RELEASE.jar（它是和事务相关的）。

1.2 JdbcTemplate 对象的创建

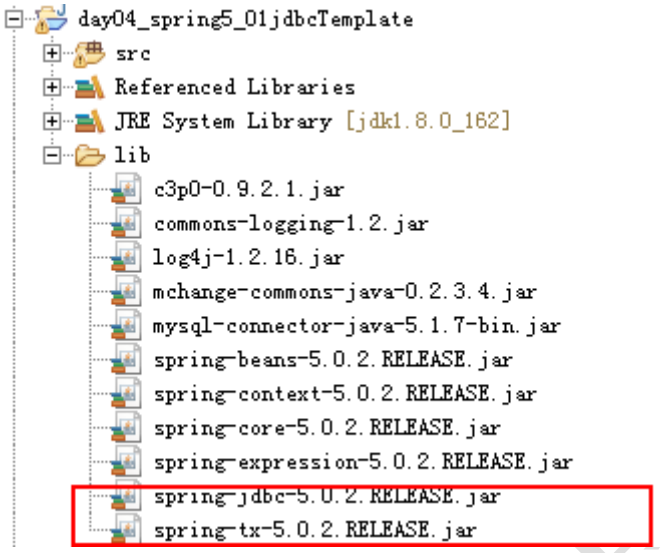
我们可以参考它的源码，来一探究竟：

```
public JdbcTemplate() {
}

public JdbcTemplate(DataSource dataSource) {
    setDataSource(dataSource);
    afterPropertiesSet();
}

public JdbcTemplate(DataSource dataSource, boolean lazyInit) {
    setDataSource(dataSource);
    setLazyInit(lazyInit);
    afterPropertiesSet();
}
```

除了默认构造函数之外，都需要提供一个数据源。既然有 set 方法，依据我们之前学过的依赖注入，我们可以在配置文件中配置这些对象。

1.3spring 中配置数据源

1.3.1 环境搭建



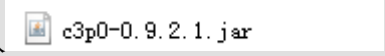
1.3.2 编写 spring 的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

</beans>
```

1.3.3 配置数据源

我们之前已经接触过了两个数据源，C3P0 和 DBCP。要想使用这两数据源都需要导入对应的 jar 包。

1.3.3.1 配置 C3P0 数据源

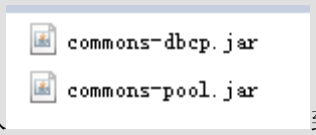
导入 

到工程的 lib 目录。在 spring 的配置文件中配置：

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
    <property name="jdbcUrl" value="jdbc:mysql:///spring_day02"></property>
    <property name="user" value="root"></property>
```

```
        <property name="password" value="1234"></property>
    </bean>
```

1.3.3.2 配置 DBCP 数据源



导入到工程的 lib 目录。在 spring 的配置文件中配置：

```
<!-- 配置数据源 -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql:// /spring_day02"></property>
    <property name="username" value="root"></property>
    <property name="password" value="1234"></property>
</bean>
```

1.3.3.3 配置 spring 内置数据源

spring 框架也提供了一个内置数据源，我们也可以使用 spring 的内置数据源，它就在 spring-jdbc-5.0.2.REEASE.jar 包中：

```
<bean                                                    id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql:///spring_day02"></property>
    <property name="username" value="root"></property>
    <property name="password" value="1234"></property>
</bean>
```

1.3.4 将数据库连接的信息配置到属性文件中：

【定义属性文件】

```
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///spring_day02
jdbc.username=root
jdbc.password=123
```

【引入外部的属性文件】

一种方式：

```
    <!-- 引入外部属性文件： -->
    <bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
```

```
        <property name="location" value="classpath:jdbc.properties"/>
       </bean>

    另一种方式：
    <context:property-placeholder location="classpath:jdbc.properties"/>
```

1.4JdbcTemplate 的增删改查操作

1.4.1 前期准备

```
    创建数据库：
    create database spring_day02;
    use spring_day02;
    创建表：
    create table account(
        id int primary key auto_increment,
        name varchar(40),
        money float
    )character set utf8 collate utf8_general_ci;
```

1.4.2 在 spring 配置文件中配置 JdbcTemplate

```
    <?xml version="1.0" encoding="UTF-8"?>
    <beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

        <!-- 配置一个数据库的操作模板：JdbcTemplate -->
        <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
            <property name="dataSource" ref="dataSource"></property>
        </bean>

        <!-- 配置数据源 -->
        <bean                                                    id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql:///spring_day02"></property>
        <property name="username" value="root"></property>
        <property name="password" value="1234"></property>
    </bean>
    </beans>
```

1.4.3 最基本使用

```
public class JdbcTemplateDemo2 {
    public static void main(String[] args) {
        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        jt.execute("insert into account(name,money)values('eee',500)");
    }
}
```

1.4.4 保存操作

```
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {

        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        //保存
        jt.update("insert into account(name,money)values(?,?)","fff",5000);
    }
}
```

1.4.5 更新操作

```
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {

        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        //修改
        jt.update("update account set money = money-? where id = ?",300,6);
    }
}
```

1.4.6 删除操作

```
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {

        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        //删除
        jt.update("delete from account where id = ?",6);
    }
}
```

1.4.7 查询所有操作

```
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {

        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        //查询所有
        List<Account> accounts = jt.query("select * from account where money > ? ",
                                          new AccountRowMapper(), 500);
        for(Account o : accounts){
            System.out.println(o);
        }
    }
}

public class AccountRowMapper implements RowMapper<Account>{
    @Override
    public Account mapRow(ResultSet rs, int rowNum) throws SQLException {
        Account account = new Account();
        account.setId(rs.getInt("id"));
        account.setName(rs.getString("name"));
        account.setMoney(rs.getFloat("money"));
        return account;
    }
```

```
}
```

1.4.8 查询一个操作

```
使用 RowMapper 的方式：常用的方式
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {

        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        //查询一个
        List<Account> as = jt.query("select * from account where id = ? ",
                                    new AccountRowMapper(), 55);
        System.out.println(as.isEmpty()?"没有结果":as.get(0));
    }
}

使用 ResultSetExtractor 的方式:不常用的方式
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {

        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
        //3.执行操作
        //查询一个
        Account account = jt.query("select * from account where id = ?",
                                   new AccountResultSetExtractor(),3);
        System.out.println(account);
    }
}
```

1.4.9 查询返回一行一列操作

```
public class JdbcTemplateDemo3 {
    public static void main(String[] args) {
        //1.获取 Spring 容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
        //2.根据 id 获取 bean 对象
```

```
            JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");
            //3.执行操作
            //查询返回一行一列：使用聚合函数，在不使用 group by 字句时，都是返回一行一列。最长用的就是分页中获取总记录条数
            Integer total = jt.queryForObject("select count(*) from account where money > ?",Integer.class,500);
            System.out.println(total);
        }
    }
```

1.5 在 dao 中使用 JdbcTemplate

1.5.1 准备实体类

```
    /**
     * 账户的实体
     */
    public class Account implements Serializable {

        private Integer id;
        private String name;
        private Float money;
        public Integer getId() {
            return id;
        }
        public void setId(Integer id) {
            this.id = id;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public Float getMoney() {
            return money;
        }
        public void setMoney(Float money) {
            this.money = money;
        }
        @Override
        public String toString() {
            return "Account [id=" + id + ", name=" + name + ", money=" + money + "]";
```

```
        }
    }
```

1.5.2 第一种方式：在 dao 中定义 JdbcTemplate

```
/**
 * 账户的接口
 */
public interface IAccountDao {

    /**
     * 根据 id 查询账户信息
     * @param id
     * @return
     */
    Account findAccountById(Integer id);

    /**
     * 根据名称查询账户信息
     * @return
     */
    Account findAccountByName(String name);

    /**
     * 更新账户信息
     * @param account
     */
    void updateAccount(Account account);
}

/**
 * 账户的持久层实现类
 * 此版本的 dao，需要给 dao 注入 JdbcTemplate
 */
public class AccountDaoImpl implements IAccountDao {

    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Override
    public Account findAccountById(Integer id) {
```

```
            List<Account> list =  jdbcTemplate.query("select * from account where id = ?
",new AccountRowMapper(),id);
            return list.isEmpty()?null:list.get(0);
        }

        @Override
        public Account findAccountByName(String name) {
            List<Account> list =  jdbcTemplate.query("select * from account where name
= ? ",new AccountRowMapper(),name);
            if(list.isEmpty()){
                return null;
            }
            if(list.size()>1){
                throw new RuntimeException("结果集不唯一，不是只有一个账户对象");
            }
            return list.get(0);
        }

        @Override
        public void updateAccount(Account account) {
            jdbcTemplate.update("update  account  set  money  =  ?  where  id  =  ?
",account.getMoney(),account.getId());
        }

    }
```

配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- 配置一个 dao -->
    <bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl">
        <!-- 注入 jdbcTemplate -->
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
    </bean>

    <!-- 配置一个数据库的操作模板：JdbcTemplate -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"></property>
    </bean>
```

```
        <!-- 配置数据源 -->
        <bean                                                    id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property                                          name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
            <property name="url" value="jdbc:mysql:///spring_day04"></property>
            <property name="username" value="root"></property>
            <property name="password" value="1234"></property>
        </bean>
    </beans>
```

思考：

此种方式有什么问题吗？

答案：

有个小问题。就是我们的 dao 有很多时，每个 dao 都有一些重复性的代码。下面就是重复代码：

```
        private JdbcTemplate jdbcTemplate;

        public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
            this.jdbcTemplate = jdbcTemplate;
        }
```

能不能把它抽取出来呢？

请看下一小节。

1.5.3 第二种方式：让 dao 继承 JdbcDaoSupport

JdbcDaoSupport 是 spring 框架为我们提供的一个类，该类中定义了一个 JdbcTemplate 对象，我们可以直接获取使用，但是要想创建该对象，需要为其提供一个数据源：具体源码如下：

```
    public abstract class JdbcDaoSupport extends DaoSupport {
        //定义对象
        private JdbcTemplate jdbcTemplate;
        //set 方法注入数据源，判断是否注入了，注入了就创建 JdbcTemplate
        public final void setDataSource(DataSource dataSource) {
       if (this.jdbcTemplate == null || dataSource != this.jdbcTemplate.getDataSource())
{               //如果提供了数据源就创建 JdbcTemplate
                this.jdbcTemplate = createJdbcTemplate(dataSource);
                initTemplateConfig();
            }
        }
        //使用数据源创建 JdcbTemplate
        protected JdbcTemplate createJdbcTemplate(DataSource dataSource) {
            return new JdbcTemplate(dataSource);
        }

        //当然，我们也可以通过注入 JdbcTemplate 对象
        public final void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
```

```
            this.jdbcTemplate = jdbcTemplate;
            initTemplateConfig();
        }
        //使用 getJdbcTmeplate 方法获取操作模板对象
        public final JdbcTemplate getJdbcTemplate() {
          return this.jdbcTemplate;
        }
```

```
    /**
     * 账户的接口
     */
    public interface IAccountDao {

        /**
         * 根据 id 查询账户信息
         * @param id
         * @return
         */
        Account findAccountById(Integer id);

        /**
         * 根据名称查询账户信息
         * @return
         */
        Account findAccountByName(String name);

        /**
         * 更新账户信息
         * @param account
         */
        void updateAccount(Account account);
    }

    /**
     * 账户的持久层实现类
     * 此版本 dao，只需要给它的父类注入一个数据源
     */
    public class AccountDaoImpl2 extends JdbcDaoSupport implements IAccountDao {

        @Override
        public Account findAccountById(Integer id) {
            //getJdbcTemplate()方法是从父类上继承下来的。
            List<Account> list = getJdbcTemplate().query("select * from account where
id = ? ",new AccountRowMapper(),id);
```

```
            return list.isEmpty()?null:list.get(0);
        }

        @Override
        public Account findAccountByName(String name) {
            //getJdbcTemplate()方法是从父类上继承下来的。
            List<Account> list =  getJdbcTemplate().query("select * from account where
name = ? ",new AccountRowMapper(),name);
            if(list.isEmpty()){
                return null;
            }
            if(list.size()>1){
                throw new RuntimeException("结果集不唯一，不是只有一个账户对象");
            }
            return list.get(0);
        }

        @Override
        public void updateAccount(Account account) {
            //getJdbcTemplate()方法是从父类上继承下来的。
            getJdbcTemplate().update("update account set money = ? where id = ?
",account.getMoney(),account.getId());
        }
    }
```

配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- 配置 dao2 -->
    <bean id="accountDao2" class="com.itheima.dao.impl.AccountDaoImpl2">
        <!-- 注入 dataSource -->
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!-- 配置数据源 -->
    <bean                                                    id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql:///spring_day04"></property>
        <property name="username" value="root"></property>
        <property name="password" value="1234"></property>
    </bean>
</beans>
```

思考：

两版 Dao 有什么区别呢？

答案：

第一种在 Dao 类中定义 JdbcTemplate 的方式，适用于所有配置方式（xml 和注解都可以）。

第二种让 Dao 继承 JdbcDaoSupport 的方式，只能用于基于 XML 的方式，注解用不了。

使用注解生成时会检查是否有