1. 基于aspectj的注解aop

1 使用注解方式实现aop操作

第一步 创建对象

```
<!-- 创建对象 -->
<bean id="book" class="cn.itcast.aop.Book"></bean>
<bean id="myBook" class="cn.itcast.aop.MyBook"></bean>
```

第二步 在spring核心配置文件中，开启aop操作

```
<!-- 开启aop操作 -->
<aop:aspectj-autoproxy></aop:aspectj-autoproxy>
```

第三步 在增强类上面使用注解完成aop操作

```
@Aspect
public class MyBook {

    //在方法上面使用注解完成增强配置
    @Before(value="execution(* cn.itcast.aop.Book.*(..))")
    public void before1() {
        System.out.println("before.............");
    }
}
```

2. Spring的jdbcTemplate操作

1 spring框架一站式框架

（1）针对javaee三层，每一层都有解决技术

（2）在dao层，使用 jdbcTemplate

2 spring对不同的持久化层技术都进行封装

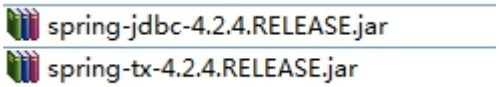
ORM持久化技术	模板类
JDBC	org.springframework.jdbc.core.JdbcTemplate
Hibernate5.0	org.springframework.orm.hibernate5.HibernateTemplate
IBatis(MyBatis)	org.springframework.orm.ibatis.SqlMapClientTemplate
JPA	org.springframework.orm.jpa.JpaTemplate

（1）jdbcTemplate对jdbc进行封装

3 jdbcTemplate使用和dbutils使用很相似，都数据库进行crud操作

2-1. 增加

1 导入jdbcTemplate使用的jar包



2 创建对象，设置数据库信息

3 创建jdbcTemplate对象，设置数据源

4 调用jdbcTemplate对象里面的方法实现操作

update(String sql, Object... args) : int - JdbcTemplate

```
@Test
public void add() {
    //设置数据库信息
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql:///spring_day03");
    dataSource.setUsername("root");
    dataSource.setPassword("root");

    //创建jdbcTemplate对象，设置数据源
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

    //调用jdbcTemplate对象里面的方法实现操作
    //创建sql语句
    String sql = "insert into user values(?,?)";
    int rows = jdbcTemplate.update(sql, "lucy","250");
    System.out.println(rows);
}
```

2-2. 修改

```java
@Test
public void update() {
    //设置数据库信息
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql:///spring_day03");
    dataSource.setUsername("root");
    dataSource.setPassword("root");

    //创建jdbcTemplate对象
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

    //调用jdbcTemplate里面的方法实现 update方法
    String sql = "update user set password=? where username=?";
    int rows = jdbcTemplate.update(sql, "1314","lucy");
    System.out.println(rows);
}
```

2-3. 删除

```java
public void delete() {
    //设置数据库信息
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql:///spring_day03");
    dataSource.setUsername("root");
    dataSource.setPassword("root");

    //创建jdbcTemplate对象
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

    //调用update方法实现删除
    String sql = "delete from user where username=?";
    int rows = jdbcTemplate.update(sql, "lucy");
    System.out.println(rows);
}
```

2-4. 查询

1 使用jdbcTemplate实现查询操作+JavaWeb阶段dbutils工具类的复习回顾

```
/*
 * QueryRunner runner = new QueryRuner(datasource);
 * 返回对象
 * runner.query(sql,new BeanHandler<User>(User.class));
 *
 * 返回list集合
 * runner.query(sql,new BeanListHander<User>(User.class))
 *
 * 1 在dbutils时候，有接口 ResultSetHandler
 * dbutils提供了针对不同的结果实现类
 *
 * 2 jdbcTemplate实现查询，有接口 RowMapper，
 * jdbcTemplate针对这个接口没有提供实现类，得到不同的类型数据需要自己进行数据封装
 *
 * */
```

2 查询具体实现

2-4-1. 查询返回某一个值

```
queryForObject(String sql, Class<T> requiredType) : T - Jdb
```

（1）第一个参数是sql语句

（2）第二个参数 返回类型的class

```
//创建jdbcTemplate对象
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

//调用方法得到记录数
String sql = "select count(*) from user";
//调用jdbcTemplate的方法
int count = jdbcTemplate.queryForObject(sql, Integer.class);
System.out.println(count);
```

Jdbc实现

```java
try {
    Class.forName("com.mysql.jdbc.Driver");
    //创建连接
    conn = DriverManager.getConnection("jdbc:mysql:///spring_day03", "root", "ro
    //编写sql语句
    String sql = "select * from user where username=?";
    //预编译sql
    psmt = conn.prepareStatement(sql);
    //设置参数值
    psmt.setString(1, "lucy");
    //执行sql
    rs = psmt.executeQuery();
    //遍历结果集
    while(rs.next()) {
        //得到返回结果值
        String username = rs.getString("username");
        String password = rs.getString("password");
        //放到user对象里面
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);

        System.out.println(user);
    }
```

2-4-2. 查询返回对象

```
queryForObject(String sql, RowMapper<T> rowMapper, Object... args) : T
```

第一个参数是sql语句

第二个参数是 RowMapper，是接口，类似于dbutils里面接口

第三个参数是 可变参数

```java
//创建jdbcTemplate对象
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

//写sql语句，根据username查询
String sql = "select * from user where username=?";
//调用jdbcTemplate的方法实现
//第二个参数是接口 RowMapper，需要自己写类实现接口，自己做数据封装
User user = jdbcTemplate.queryForObject(sql, new MyRowMapper(), "mary");
System.out.println(user);
```

```
class MyRowMapper implements RowMapper<User> {

    @Override
    public User mapRow(ResultSet rs, int num) throws SQLException
        // 1 从结果集里面把数据得到
        String username = rs.getString("username");
        String password = rs.getString("password");

        // 2 把得到数据封装到对象里面
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);

        return user;
    }

}
```

2-4-3. 查询返回list集合

```
query(String sql, RowMapper<T> rowMapper, Object... args) : List<T>
```

（1） sql语句

（2） RowMapper接口，自己写类实现数据封装

（3） 可变参数

```
//创建jdbcTemplate对象
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

//写sql语句
String sql = "select * from user";
//调用jdbcTemplate的方法实现
List<User> list = jdbcTemplate query(sql,new MyRowMapper());

System.out.println(list);
```

3. Spring配置连接池和dao使用jdbcTemplate

3-1. spring配置c3p0连接池

第一步 导入jar包

第二步 创建spring配置文件，配置连接池

```
ComboPooledDataSource dataSource = new ComboPooledDataSource();
dataSource.setDriverClass("com.mysql.jdbc.Driver");
dataSource.setJdbcUrl("jdbc:mysql:///spring_day03");
dataSource.setUser("root");
dataSource.setPassword("root");
```

（1）把代码在配置文件中进行配置

```
<!-- 配置c3p0连接池 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <!-- 注入属性值 -->
    <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
    <property name="jdbcUrl" value="jdbc:mysql:///spring_day03"></property>
    <property name="user" value="root"></property>
    <property name="password" value="root"></property>
</bean>
```

3-2. dao使用jdbcTemplate

（1）创建service和dao，配置service和dao对象，在service注入dao对象

```
<!-- 创建service和dao对象，在service注入dao对象 -->
<bean id="userService" class="cn.itcast.c3p0.UserService">
    <!-- 注入dao对象 -->
    <property name="userDao" ref="userDao"></property>
</bean>

<bean id="userDao" class="cn.itcast.c3p0.UserDao">
```

（2）创建jdbcTemplate对象，把模板对象注入到dao里面

```
//得到JdbcTemplate对象
private JdbcTemplate jdbcTemplate;
public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}
```

```
<bean id="userDao" class="cn.itcast.c3p0.UserDao">
    <!-- 注入jdbcTemplate对象 -->
    <property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
<!-- 创建jdbcTemplate对象 -->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
```

（3）在jdbcTemplate对象里面注入dataSource

```java
private DataSource dataSource;

private SQLExceptionTranslator exceptionTranslator;

private boolean lazyInit = true;


/**
 * Set the JDBC DataSource to obtain connections from.
 */
public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
}
```

```xml
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <!-- 把dataSource传递到模板对象里面 -->
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

4. Spring的事务管理

4-1. 事务概念

1 什么事务

2 事务特性

3 不考虑隔离性产生读问题

4 解决读问题

（1）设置隔离级别

4-2. Spring事务管理api

1 spring事务管理两种方式

第一种 编程式事务管理（不用）

第二种 声明式事务管理

（1） 基于xml配置文件实现

（2） 基于注解实现

2 spring事务管理的api介绍

接口

PlatformTransactionManager

事务管理器

（1）spring针对不同的dao层框架，提供接口不同的实现类

事务	说明
org.springframework.jdbc.datasource.DataSourceTransactionManager	使用Spring JDBC或iBatis 进行持久化数据时使用
org.springframework.orm.hibernate5.HibernateTransactionManager	使用Hibernate5.0版本进行持久化数据时使用

（2）首先 配置事务管理器

4-3. 搭建转账环境

1 创建数据库表，添加数据

id	username	salary
1	小王	10000
2	小马	10000

2 创建service和dao类，完成注入关系

```
<bean id="ordersService" class="cn.itcast.service.OrdersService">
    <property name="ordersDao" ref="ordersDao"></property>
</bean>
<bean id="ordersDao" class="cn.itcast.dao.OrdersDao">
    <property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

（1）service层又叫业务逻辑层

（2）dao层，单纯对数据库操作层，在dao层不添加业务

（3）需求：小王 转账 1000 给 小马

- 小王少1000
- 小马多1000

```java
/*
 * 做对数据库操作的方法，不写业务操作
 * */
//小王少钱的方法
public void lessMoney() {
    String sql = "update account set salary=salary-? where username=?";
    jdbcTemplate.update(sql, 1000,"小王");
}

//小马多钱的方法
public void moreMoney() {
    String sql = "update account set salary=salary+? where username=?";
    jdbcTemplate.update(sql, 1000,"小马");
}
```

```java
//调用dao的方法
//业务逻辑层，写转账业务
public void accountMoney() {
    //小王少1000
    ordersDao.lessMoney();

    //小马多1000
    ordersDao.moreMoney();
}
```

3 产生问题：

（1）如果小王少了1000之后，出现异常，小马不会多1000，钱丢失了

4 解决：

（1）添加事务解决，出现异常进行回滚操作

4-4. 声明式事务管理（xml配置)

1 配置文件方式使用aop思想配置

第一步 配置事务管理器

```xml
<!-- 第一步配置事务管理器 -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager
    <!-- 注入dataSource -->
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

第二步 配置事务增强

```
<!-- 第二步 配置事务增强 -->
<tx:advice id="txadvice" transaction-manager="transactionManager">
    <!-- 做事务操作 -->
    <tx:attributes>
        <!-- 设置进行事务操作的方法匹配规则 -->
        <tx:method name="account*" propagation="REQUIRED"/>
        <!-- <tx:method name="insert*" /> -->
    </tx:attributes>
</tx:advice>
```

第三步 配置切面

```
<!-- 第三步 配置切面 -->
<aop:config>
    <!-- 切入点 -->
    <aop:pointcut expression="execution(* cn.itcast.service.OrdersService.*(..))" id="pointcut
    <!-- 切面 -->
    <aop:advisor advice-ref="txadvice" pointcut-ref="pointcut1"/>
</aop:config>
```

4-5. 声明式事务管理（注解）

第一步 配置事务管理器

```
<!-- 第一步 配置事务管理器 -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionM
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

第二步 配置事务注解

```
<!-- 第二步 开启事务注解 -->
<tx:annotation-driven transaction-manager="transactionManager"/>
```

第三步 在要使用事务的方法所在**类**上面添加注解

```
@Transactional
public class OrdersService {
```