

Spring学习日志三

一、Spring整合JDBC

spring提供了很多模板整合Dao技术

ORM持久化技术	模板类
JDBC	org.springframework.jdbc.core.JdbcTemplate
Hibernate3.0	org.springframework.orm.hibernate3.HibernateTemplate
IBatis(MyBatis)	org.springframework.orm.ibatis.SqlMapClientTemplate
JPA	org.springframework.orm.jpa.JpaTemplate

现在学的是JDBC的整合。

现说明本文所需要导入的包：



1.1 介绍JdbcTemplate核心API

1.1.1 基本介绍

```

8 //但是这些东西都可以看作是spring的一个bean而已，是可以配置到配置文件中去的，其属性，都是可以依靠依赖注入注入的
9 @test
10 public void testDBCP(){ //很明显，这是set型注入
11     //1 创建数据源（连接池） dbcp
12     BasicDataSource dataSource = new BasicDataSource(); //其实就是DBCP连接池当初的基本操作
13     // * 基本4项
14     dataSource.setDriverClassName("com.mysql.jdbc.Driver");
15     dataSource.setUrl("jdbc:mysql://localhost:3306/ee19_spring_day02");
16     dataSource.setUsername("root");
17     dataSource.setPassword("201805");
18
19     //2 创建模板
20     JdbcTemplate jdbcTemplate = new JdbcTemplate();
21     jdbcTemplate.setDataSource(dataSource); //用spring封装的JdbcTemplate模板类来执行sql语句
22                                           //只不过不用QueryRunner执行sql语句了而已
23
24     //3 通过api操作
25     jdbcTemplate.update("insert into t_user(username,password) values(?,?)", "tom", "998");
26 }
27 }

```

1.1.2 详细介绍

1.2 基于DBCP整合Dao层

1.2.1 封装一个实体类

```

3 public class User {
4     // create table t_user(
5     //     id int primary key auto_increment,
6     //     username varchar(50),
7     //     password varchar(32)
8     // );
9     //
10    // insert into t_user(username,password) values('jack','1234');
11    // insert into t_user(username,password) values('rose','5678');
12
13    private Integer id;
14    private String username;
15    private String password; //其余没有必要写

```

1.2.2 写Dao层

```

1 package cn.scct.c_dbcp;
2
3 import org.springframework.jdbc.core.JdbcTemplate;
4
5
6 public class UserDao {
7     private JdbcTemplate jdbcTemplate;
8
9     //Dao层嘛，要执行sql语句，要用这个东西执行呀，自然要作
10    public JdbcTemplate getJdbcTemplate() {
11        return jdbcTemplate; //为Dao层的成员变量最好，注意要一个DataSource，这里没
12    } //给，特依靠属性注入
13
14    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
15        this.jdbcTemplate = jdbcTemplate;
16    }
17
18    public void update(User user) { //执行sql语句的方法
19        String sql="update t_user set username=?,password=? where id=?";
20        Object[] args={user.getUsername(),user.getPassword(),user.getId()};
21        jdbcTemplate.update(sql, args);
22    }
23
24 }

```

1.2.3 注册Dao层

```

10 <!-- 0.将连接池放入spring容器 -->
11 <bean name="dataSource" class="org.apache.commons.dbcp.BasicDataSource" >
12     <property name="driverClassName" value="com.mysql.jdbc.Driver" ></property>
13     <property name="url" value="jdbc:mysql://localhost:3306/ee19_spring_day02" ></property>
14     <property name="username" value="root" ></property>
15     <property name="password" value="201805" ></property>
16 </bean>
17 //所以还要注册连接池属性，注意这里用的是DBCP连接池，不同的连接池的注入属性不一样
18 <!-- 1.将JdbcTemplate放入spring容器 -->
19 <bean name="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate" >
20     <!-- 看纯粹的实现知道需要设置数据源 --> //注册jdbcTemplate对象，必须注入连接池属性
21     <property name="dataSource" ref="dataSource" ></property>
22 </bean>
23
24 <!-- 2.将UserDao放入spring容器 --> //思路：首先，我们需要注册UserDao对象
25 <bean name="userDao" class="cn.scct.c_dbcp.UserDao" >
26     <property name="jdbcTemplate" ref="jdbcTemplate" ></property>
27 </bean>
28 //然后必须注入它的jdbcTemplate属性，所以必须注册jdbcTemplate

```

1.2.4 测试

```

10 public class TestDBCP {
11     @Test
12     public void test01() {
13         User user=new User();
14         user.setId(1);
15         user.setUsername("鬼鬼");
16         user.setPassword("1111"); //这里的对象只是传入Dao层，作为操作sql语句的参数而已，不得不自己创建
17         String xmlPath="cn/scct/c_dbcp/applicationContext.xml";
18         ApplicationContext applicationContext=new ClassPathXmlApplicationContext(xmlPath);
19         UserDao userDao = (UserDao) applicationContext.getBean("userDao"); //获取Dao层的对象 执行sql语句
20         userDao.update(user);
21     }
22 }

```

//有人会这样想到，不是所有对象都该注册到spring中去吗？这是
在是个误解，因为spring中注册的是经常使用的，属性不变的对
象，需要变的对象注册它干嘛？控制反转有何意义？有一百个对象
需要你全都去注册吗？

结果就不演示了，没啥必要，这个内容就是更改一下对应数据库某个表ID为1的数据而已。

1.3 基于C3P0整合Dao层

仅仅是配置不一样而已。

```
10 <!-- 0.将连接池放入spring容器 -->
11 <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
12     <property name="driverClass" value="com.mysql.jdbc.Driver" ></property>
13     <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ee19_spring_day02" ></property>
14     <property name="user" value="root" ></property>
15     <property name="password" value="201805" ></property>
16 </bean>
17
18 <!-- 1.将JdbcTemplate放入spring容器 -->
19 <bean name="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate" >
20     <!-- 看纯粹的实现知道需要设置数据源 -->
21     <property name="dataSource" ref="dataSource" ></property>
22 </bean>
23
24 <!-- 2.将UserDao放入spring容器 -->
25 <bean name="userDao" class="cn.scct.d_c3p0.UserDao" >
26     <property name="jdbcTemplate" ref="jdbcTemplate" ></property>
27 </bean>
```

仅仅是C3P0的属性名称不一样罢了

1.4 Dao层继承JdbcSupport

```
8 public class UserDao extends JdbcDaoSupport {
9
10     //Dao层继承了JdbcDaoSupport之后，不用自己搞一个JdbcTemplate属性，它本来就有了
11     public void update(User user){
12         String sql="update t_user set username=?,password=? where id=?";
13         Object[] args={user.getUsername(),user.getPassword(),user.getId()};
14         this.getJdbcTemplate().update(sql, args);
15     }
16     //用this.getJdbcTemplate()获得JdbcTemplate对象
17 }
```

随之而来的便宜就是不用再Dao层注入JdbcTemplate属性了。

```
10 <!-- 0.将连接池放入spring容器 -->
11 <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
12     <property name="driverClass" value="com.mysql.jdbc.Driver" ></property>
13     <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ee19_spring_day02" ></property>
14     <property name="user" value="root" ></property>
15     <property name="password" value="201805" ></property>
16 </bean>
17
18 <!-- 1.将UserDao放入spring容器 -->
19 <bean name="userDao" class="cn.scct.e_jdbcdaosupport.UserDao" >
20     <property name="dataSource" ref="dataSource" ></property>
21 </bean>
```

//直接就是连接池，不需要注入jdbcTemplate属性

1.5 读properties文件

这是我们jdbc的properties文件，用来存放数据库连接的一些信息

```
1 jdbc.driverClass=com.mysql.jdbc.Driver
2 jdbc.jdbcUrl=jdbc:mysql:///ee19_spring_day02
3 jdbc.user=root
4 jdbc.password=201805
```

//properties文件很简单，name=value,=左右都不允许有空格

在spring中使用该properties文件

```
10 <context:property-placeholder location="classpath:cn/scct/f_properties/jdbc.properties"/>
11
12 <!-- 0.将连接池放入spring容器 --> //必须有这个配置，表示导入了properties文件
13 <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
14     <property name="driverClass" value="${jdbc.driverClass}" ></property>
15     <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
16     <property name="user" value="${jdbc.user}" ></property>
17     <property name="password" value="${jdbc.password}" ></property>
18 </bean> //取值直接就是${properties文件中的键名}
19
20 <!-- 1.将UserDao放入spring容器 -->
21 <bean name="userDao" class="cn.scct.f_properties.UserDao" >
22     <property name="dataSource" ref="dataSource" ></property>
23 </bean>
```

二、Spring中的事务管理

事务操作对象

因为在不同平台,操作事务的代码各不相同.spring提供了一个接口

PlatformTransactionManager 接口

DataSourceTransactionManager JDBC

HibernateTransitionmanager Hibernate

注意:在spring中玩事务管理.最为核心的对象就是TransactionManager对象

传播行为	含义
PROPAGATION_REQUIRED	表示当前方法必须运行在事务中。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新的事务 必须要事务 可以用调用者的事务 常用
PROPAGATION_SUPPORTS	表示当前方法不需要事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行 可有可无，无所谓，有的话就在事务中运行
PROPAGATION_MANDATORY	表示该方法必须在事务中运行，如果当前事务不存在，则会抛出一个异常 没有不行，没有抛异常
PROPAGATION_REQUIRED_NEW	表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前事务会被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager 必须拥有自己的事务，不用调用者开的事务
PROPAGATION_NOT_SUPPORTED	表示该方法不应该运行在事务中。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager 不需要事务，给我也不要
PROPAGATION_NEVER	表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常
PROPAGATION_NESTED	表示如果当前已经存在一个事务，那么该方法将会在该事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在，那么其行为与PROPAGATION_REQUIRED一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务

本节需要导入的包：

lib

- com.springsource.com.mchange.v2.c3p0-0.9.1.2.jar **c3p0连接池包**
- com.springsource.org.aopalliance-1.0.0.jar **aop联盟包**
- com.springsource.org.apache.commons.dbcp-1.2.2.cgi.jar **虽然本节用的是C3P0连接池，但如果要用DBCP连接池，则必须要用这两个包**
- com.springsource.org.apache.commons.logging-1.1.1.jar
- com.springsource.org.apache.commons.pool-1.5.3.jar
- com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar
- mysql-connector-java-5.1.7-bin.jar **驱动包**
- spring-aop-4.2.4.RELEASE.jar
- spring-aspects-4.2.4.RELEASE.jar
- spring-beans-4.2.4.RELEASE.jar
- spring-context-4.2.4.RELEASE.jar
- spring-core-4.2.4.RELEASE.jar
- spring-expression-4.2.4.RELEASE.jar
- spring-jdbc-4.2.4.RELEASE.jar
- spring-test-4.2.4.RELEASE.jar
- spring-tx-4.2.4.RELEASE.jar **事务**

最基本的包 玩aop必备

jdbc开发 测试

本节需要导入的约束：

2. 导入新的约束(tx)

xsi	http://www.w3.org/200...	
<no pref...	http://www.springfram...	http://www.springfram...
context	http://www.springfram...	http://www.springfram...
aop	http://www.springfram...	http://www.springfram...
tx	http://www.springfram...	http://www.springfram...

beans: 最基本
context: 读取properties配置
aop: 配置aop
tx: 配置事务通知

2.1 编码方式管理(不推荐)

2.1.1 编写dao层

```
1 package cn.scct.dao;
2
3 public interface AccountDao {
4     //一个接口，一个实现类，标配
5     //加钱
6     void increaseMoney(Integer id, Double money);
7     //减钱
8     void decreaseMoney(Integer id, Double money);
9 }
10
11 public class AccountDaoImpl extends JdbcDaoSupport implements AccountDao {
12     //实现类
13     @Override
14     public void increaseMoney(Integer id, Double money) {
15         //继承JdbcDaoSupport就可以免注入JdbcTemplate了，这点上面已经提到
16         getJdbcTemplate().update("update t_account set money = money+? where id = ? ", money, id);
17     }
18     @Override
19     public void decreaseMoney(Integer id, Double money) {
20         getJdbcTemplate().update("update t_account set money = money-? where id = ? ", money, id);
21     }
22 }
```

2.1.2 编写service层

```

1 package cn.scct.service;
2
3 public interface AccountService {
4     //转账方法
5     void transfer(Integer from,Integer to,Double money);
6 }

```

```

public class AccountServiceImpl implements AccountService {
    private AccountDao ad ; //这个自然是要
    private TransactionTemplate tt; //但不要忽略了事务模板对象
    public void setTt(TransactionTemplate tt) {
        this.tt = tt;
    }
    public void setAd(AccountDao ad) {
        this.ad = ad;
    }

    @Override
    public void transfer(final Integer from,final Integer to,final Double money) {

        tt.execute(new TransactionCallbackWithoutResult() {
            @Override
            protected void doInTransactionWithoutResult(TransactionStatus arg0) {
                //减钱
                ad.decreaseMoney(from, money);
                int i = 1/0;
                //加钱
                ad.increaseMoney(to, money);
            }
        });
    }
}

```

2.1.3 编写配置文件

```

47 <!-- 0.将连接池放入spring容器 -->
48 <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
49     <property name="driverClass" value="${jdbc.driverClass}" ></property>
50     <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
51     <property name="user" value="${jdbc.user}" ></property>
52     <property name="password" value="${jdbc.password}" ></property>
53 </bean>
54
55 <!-- 1.将AccountDao放入spring容器 -->
56 <bean name="accountDao" class="cn.scct.dao.AccountDaoImpl" >
57     <property name="dataSource" ref="dataSource" ></property>
58 </bean>
59
60 <!-- 2.将AccountService放入spring容器 -->
61 <bean name="accountService" class="cn.scct.service.AccountServiceImpl" >
62     <property name="ad" ref="accountDao" ></property>
63     <property name="tt" ref="transactionTemplate" ></property>
64 </bean>

```

```

4 <context:property-placeholder location="classpath:jdbc.properties"/> //连接池的配置文件引入
5
6 <!-- 事务核心管理器 封装了所有事务操作，依赖于连接池 --> //事务核心管理器
7 <bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
8     <property name="dataSource" ref="dataSource" ></property>
9 </bean>
10 <!-- 事务模板对象 --> //事务模板对象
11 <bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate" >
12     <property name="transactionManager" ref="transactionManager" ></property>
13 </bean>

```

2.1.4 测试


```

13 @RunWith(SpringJUnit4ClassRunner.class)
14 @ContextConfiguration("classpath:applicationContext.xml")
15 public class Demo {
16     @Resource(name="accountService")
17     private AccountService as;
18     @Test
19     public void test1(){
20         as.transfer(2, 1, 666d);
21     }
22 }
23 }

```

2.2 xml配置方式(推荐)

2.2.1 service层

```

10 public class AccountServiceImpl implements AccountService {
11     private AccountDao ad ;
12
13     public void setAd(AccountDao ad) {
14         this.ad = ad;
15     }
16     // private TransactionTemplate tt;
17     // public void setTt(TransactionTemplate tt) {
18     //     this.tt = tt;
19     // }
20     @Override
21     @Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readonly=false)
22     public void transfer(final Integer from,final Integer to,final Double money) {
23         //减钱
24         ad.decreaseMoney(from, money);
25         // int i = 1/0;
26         //加钱
27         ad.increaseMoney(to, money);
28     }

```

2.2.2 配置文件更改

```

4 <context:property-placeholder location="classpath:jdbc.properties"/> //连接池配置文件
5
6 <!-- 事务核心管理器,封装了所有事务操作, 依赖于连接池 --> //切莫忘了事务核心管理器, 事务管理也需要它
7 <bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
8   <property name="dataSource" ref="dataSource" ></property>
9 </bean>
10
11 <!-- 0. 将连接池放入spring容器 -->
12 <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
13   <property name="driverClass" value="${jdbc.driverClass}" ></property>
14   <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
15   <property name="user" value="${jdbc.user}" ></property>
16   <property name="password" value="${jdbc.password}" ></property>
17 </bean>
18
19 <!-- 1. 将AccountDao放入spring容器 -->
20 <bean name="accountDao" class="cn.scct.dao.AccountDaoImpl" >
21   <property name="dataSource" ref="dataSource" ></property>
22 </bean>
23
24 <!-- 2. 将AccountService放入spring容器 -->
25 <bean name="accountService" class="cn.scct.service.AccountServiceImpl" >
26   <property name="ad" ref="accountDao" ></property>
27   <!-- <property name="tt" ref="transactionTemplate" ></property> -->
28 </bean>
29
30 <tx:advice id="txAdvice" transaction-manager="transactionManager" > //需要事务管理器
31   <tx:attributes>
32     <!-- 以方法为单位, 指定方法应用什么事务属性
33     isolation: 隔离级别
34     propagation: 传播行为
35     read-only: 是否只读
36     -->
37     <tx:method name="save*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
38     <tx:method name="persist*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
39     <tx:method name="update*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
40     <tx:method name="modify*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
41     <tx:method name="delete*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
42     <tx:method name="remove*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
43     <tx:method name="get*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true" />
44     <tx:method name="find*" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="true" />
45     <tx:method name="transfer" isolation="REPEATABLE_READ" propagation="REQUIRED" read-only="false" />
46   </tx:attributes>
47 </tx:advice>
48
49 <!-- 配置织入 -->
50 <aop:config >
51   <!-- 配置切入点表达式 -->
52   <aop:pointcut expression="execution(* cn.scct.service.*ServiceImpl.*(..))" id="txPc"/>
53   <!-- 配置切面: 通知+切入点
54   advice-ref: 通知的名称
55   pointcut-ref: 切点的名称
56   -->
57   <aop:advisor advice-ref="txAdvice" pointcut-ref="txPc" />
58 </aop:config>

```

2.3 注解配置(推荐, 最简洁)

2.3.1 service层

```

9  @Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readonly=true)
10 public class AccountServiceImpl implements AccountService {
11     private AccountDao ad ;
12                                     //在类上加事务控制，表示对所有方法加同样的事务控制
13     public void setAd(AccountDao ad) {
14         this.ad = ad;
15     }
16     // private TransactionTemplate tt;
17     // public void setTt(TransactionTemplate tt) { 注解方式
18     //     this.tt = tt;
19     // }
20     @Override //在方法上设置事务属性，表示要给它加事务控制可以覆盖类的事务控制
21     @Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readonly=false)
22     public void transfer(final Integer from,final Integer to,final Double money) {
23         // 减钱
24         ad.decreaseMoney(from, money);
25         // int i = 1/0;
26         // 加钱
27         ad.increaseMoney(to, money);
28     }

```

2.3.2 xml配置文件更改

```

4  <context:property-placeholder location="classpath:jdbc.properties"/>
5
6  <!-- 事务核心管理器，封装了所有事务操作，依赖于连接池 -->
7  <bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
8      <property name="dataSource" ref="dataSource" ></property>
9  </bean>
10                                     还有事务核心管理器，无论如何不能丢
11 <!-- 事务模板对象 -->
12 <!-- <bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate" >
13     <property name="transactionManager" ref="transactionManager" ></property>
14 </bean>
15 -->
16 <!-- 开启使用注解管理aop事务 -->
17 <tx:annotation-driven/>
18                                     只需要添加这句话
19 <!-- 0. 将连接池放入spring容器 -->
20 <bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
21     <property name="driverClass" value="${jdbc.driverClass}" ></property>
22     <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
23     <property name="user" value="${jdbc.user}" ></property>
24     <property name="password" value="${jdbc.password}" ></property>
25 </bean>
26 <!-- 1. 将AccountDao放入spring容器 -->
27 <bean name="accountDao" class="cn.scct.dao.AccountDaoImpl" >
28     <property name="dataSource" ref="dataSource" ></property>
29 </bean>
30
31 <!-- 2. 将AccountService放入spring容器 -->
32 <bean name="accountService" class="cn.scct.service.AccountServiceImpl" >
33     <property name="ad" ref="accountDao" ></property>
34     <!-- <property name="tt" ref="transactionTemplate" ></property> -->
35 </bean>

```