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

```java
//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
@Test
public void testDBCP(){     //很明显，这是set型注入
    //1 创建数据源（连接池）dbcp
            BasicDataSource dataSource = new BasicDataSource();
            // * 基本4项                                          //其实就是DBCP连接池当初的基本操作
            dataSource.setDriverClassName("com.mysql.jdbc.Driver");
            dataSource.setUrl("jdbc:mysql://localhost:3306/ee19_spring_day02");
            dataSource.setUsername("root");
            dataSource.setPassword("201805");


            //2 创建模板
            JdbcTemplate jdbcTemplate = new JdbcTemplate();
            jdbcTemplate.setDataSource(dataSource);   //用spring封装的JdbcTemplate模板类来执行sql语句
                                                      //只不过不用QueryRunner执行sql语句了而已

            //3 通过api操作
            jdbcTemplate.update("insert into t_user(username,password) values(?,?);", "tom","998");
}

}
```

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

```java
public class User {
//  create table t_user(
//            id int primary key auto_increment,
//            username varchar(50),
//            password varchar(32)
//          );
//
//          insert into t_user(username,password) values('jack','1234');
//          insert into t_user(username,password) values('rose','5678');

    private Integer id;
    private String username;
    private String  password;  //其余没有必要写
```

1.2.2 写Dao层

```java
package cn.scct.c_dbcp;

import org.springframework.jdbc.core.JdbcTemplate;

public class UserDao {
    private JdbcTemplate jdbcTemplate;

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }

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

    public void update(User user){  //执行sql语句的方法
        String sql="update t_user set username=?,password=? where id=?";
        Object[] args={user.getUsername(),user.getPassword(),user.getId()};
        jdbcTemplate.update(sql, args);
    }

}
```

//Dao层嘛，要执行sql语句，要用这个东西执行呀，自然要作为Dao层的成员变量最好，注意要一个DataSource，这里没给，特依靠属性注入

1.2.3 注册Dao层

```xml
<!-- 0.将连接池放入spring容器 -->
<bean name="dataSource" class="org.apache.commons.dbcp.BasicDataSource" >
    <property name="driverClassName" value="com.mysql.jdbc.Driver" ></property>
    <property name="url" value="jdbc:mysql://localhost:3306/ee19_spring_day02" ></property>
    <property name="username" value="root" ></property>
    <property name="password" value="201805" ></property>
</bean>

<!-- 1.将JDBCTemplate放入spring容器 -->
<bean name="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate" >
    <!-- 看纯粹的实现知道需要设置数据源 -->
    <property name="dataSource" ref="dataSource" ></property>
</bean>

<!-- 2.将UserDao放入spring容器 -->
<bean name="userDao" class="cn.scct.c_dbcp.UserDao" >
    <property name="jdbcTemplate" ref="jdbcTemplate" ></property>
</bean>
```

//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

//注册jdbcTemplate对象，必须注入连接池属性

//思路：首先，我们需要注册UserDao对象

//然后必须注入它的jdbcTemplate属性,所以必须注册jdbcTemplate

1.2.4 测试

```java
public class TestDBCP {
    @Test
    public  void test01() {
    User user=new User();
    user.setId(1);
    user.setUsername("鬼鬼");
    user.setPassword("1111");
    String xmlPath="cn/scct/c_dbcp/applicationContext.xml";
    ApplicationContext applicationContext=new ClassPathXmlApplicationContext(xmlPath);
    UserDao userDao = (UserDao) applicationContext.getBean("userDao");
    userDao.update(user);
    }
}
```

//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都去注册吗？

//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

//获取Dao层的对象 执行sql语句

结果就不演示了，没有啥必要，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。

```xml
<!-- 0.将连接池放入spring容器 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
    <property name="driverClass" value="com.mysql.jdbc.Driver" ></property>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ee19_spring_day02" ></property>
    <property name="user" value="root" ></property>
    <property name="password" value="201805" ></property>
</bean>

<!-- 1.将JDBCTemplate放入spring容器 -->
<bean name="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate" >
    <!-- 看纯粹的实现知道需要设置数据源 -->
    <property name="dataSource" ref="dataSource" ></property>
</bean>

<!-- 2.将UserDao放入spring容器 -->
<bean name="userDao" class="cn.scct.d_c3p0.UserDao" >
    <property name="jdbcTemplate" ref="jdbcTemplate" ></property>
</bean>
```

仅仅是C3P0的属性名称不一样罢了

1.4 Dao层继承JdbcSupport

```java
public class UserDao extends JdbcDaoSupport {

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

随之而来的便宜就是不用再Dao层注入JdbcTemplate属性了。

```xml
<!-- 0.将连接池放入spring容器 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
    <property name="driverClass" value="com.mysql.jdbc.Driver" ></property>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ee19_spring_day02" ></property>
    <property name="user" value="root" ></property>
    <property name="password" value="201805" ></property>
</bean>

<!-- 1.将UserDao放入spring容器 -->
<bean name="userDao" class="cn.scct.e_jdbcdaosupport.UserDao" >
    <property name="dataSource" ref="dataSource" ></property>
</bean>
```

//直接就是连接池，不需要注入jdbcTemplate属性

1.5 读properties文件

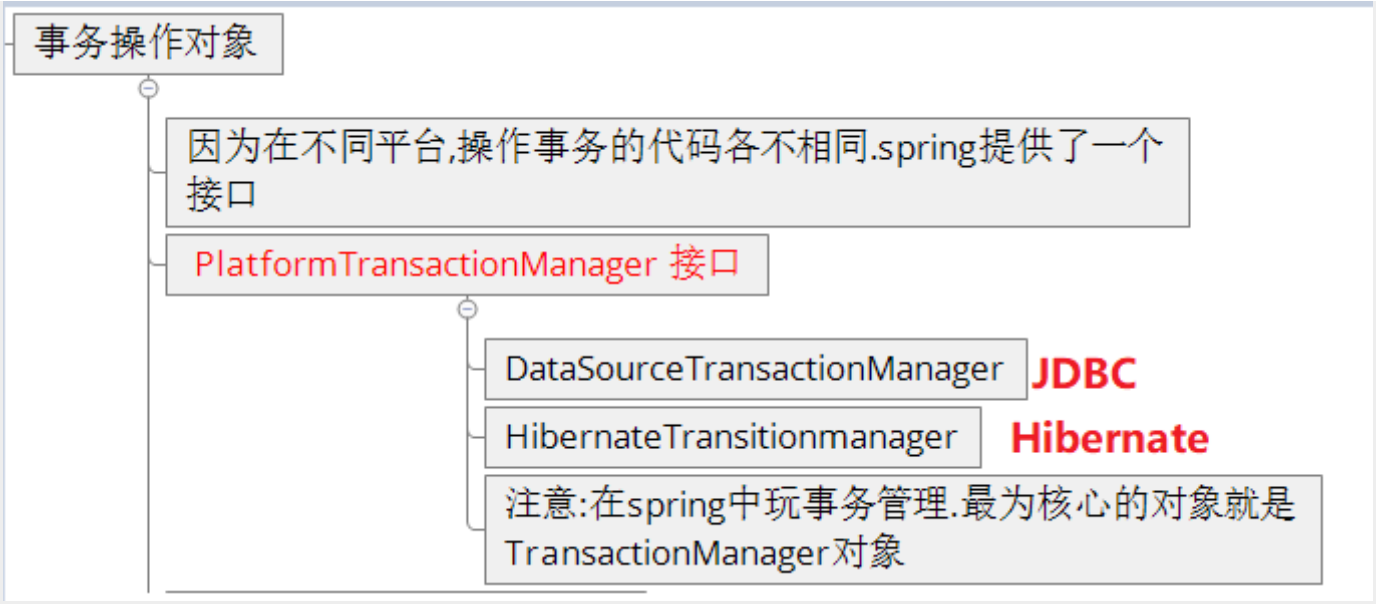
这是我们jdbc的properties文件，用来存放数据库连接的一些信息

```
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.jdbcUrl=jdbc:mysql:///ee19_spring_day02
jdbc.user=root
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

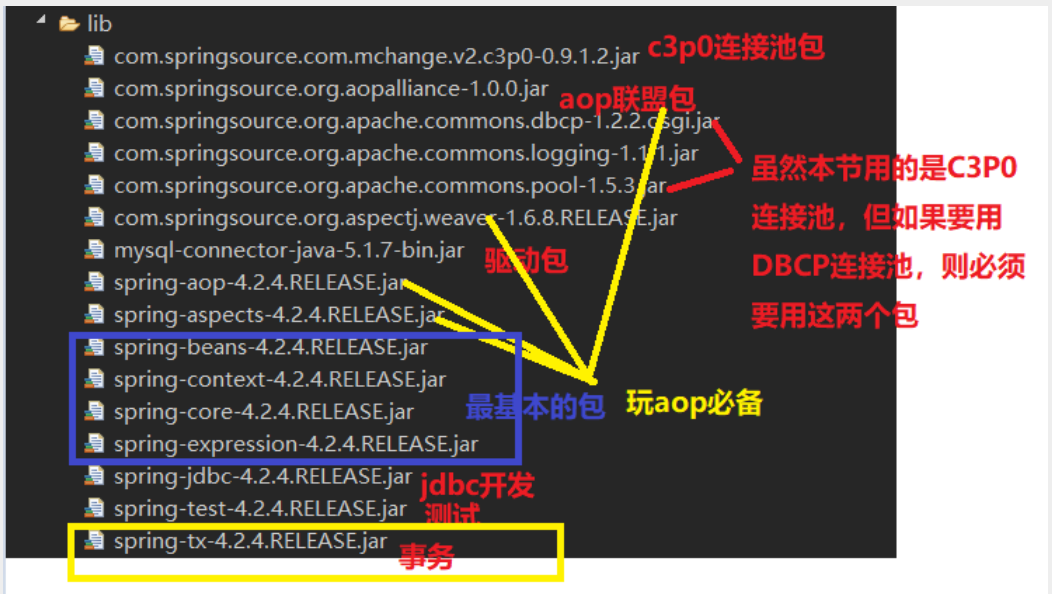
```
<context:property-placeholder location="classpath:cn/scct/f_properties/jdbc.properties"/>

                                        //必须有这个配置，表示导入了properties文件
<!-- 0.将连接池放入spring容器 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
    <property name="driverClass" value="${jdbc.driverClass}" ></property>
    <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
    <property name="user" value="${jdbc.user}" ></property>
    <property name="password" value="${jdbc.password}" ></property>
</bean>
                                        //取值直接就是${properties文件中的键名}

<!-- 1.将UserDao放入spring容器 -->
<bean name="userDao" class="cn.scct.f_properties.UserDao" >
    <property name="dataSource" ref="dataSource" ></property>
</bean>
```

二、Spring中的事务管理



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，那么该方法将会在嵌套事务中运行。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:



本节需要导入的约束：

2.1 编码方式管理(不推荐)

2.1.1 编写dao层

```java
package cn.scct.dao;

public interface AccountDao {
                //一个接口，一个实现类，标配
    //加钱
    void increaseMoney(Integer id,Double money);
    //减钱
    void decreaseMoney(Integer id,Double money);

}
```

```java
public class AccountDaoImpl extends JdbcDaoSupport implements AccountDao  {

    @Override                    //实现类
    public void increaseMoney(Integer id, Double money) {

        getJdbcTemplate().update("update t_account set money = money+? where id = ? ", money,id);

    }

    @Override
    public void decreaseMoney(Integer id, Double money) {

        getJdbcTemplate().update("update t_account set money = money-? where id = ? ", money,id);
    }

}
```

继承JdbcDaoSupport就可以免注入JdbcTemplate了，这点上面已经提到了

2.1.2 编写service层

```java
package cn.scct.service;

public interface AccountService {
    //转账方法
        void transfer(Integer from,Integer to,Double money);
}
```

```java
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);
            }
        });


    }
```

//调用事务模板对象执行excute()方法

2.1.3 编写配置文件

```xml
<!-- 0.将连接池放入spring容器 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
    <property name="driverClass" value="${jdbc.driverClass}" ></property>
    <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
    <property name="user" value="${jdbc.user}" ></property>
    <property name="password" value="${jdbc.password}" ></property>
</bean>

<!-- 1.将AccountDao放入spring容器 -->
<bean name="accountDao" class="cn.scct.dao.AccountDaoImpl" >
    <property name="dataSource" ref="dataSource" ></property>
</bean>

<!-- 2.将AccountService放入spring容器 -->
<bean name="accountService" class="cn.scct.service.AccountServiceImpl" >
    <property name="ad" ref="accountDao" ></property>
    <property name="tt" ref="transactionTemplate" ></property>
</bean>
```

```xml
<context:property-placeholder location="classpath:jdbc.properties"/> //连接池的配置文件引入

<!-- 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
<bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
    <property name="dataSource" ref="dataSource" ></property>
</bean>
<!-- 事务模板对象 -->
<bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate" >
    <property name="transactionManager" ref="transactionManager" ></property>
</bean>
```

//事务模板对象

2.1.4 测试

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Demo {
    @Resource(name="accountService")
    private AccountService as;
   @Test
    public void test1(){
        as.transfer(2, 1, 666d);

    }
}
```

2.2 xml配置方式(推荐)

2.2.1 service层

```java
public class AccountServiceImpl implements AccountService  {
    private AccountDao ad ;

    public void setAd(AccountDao ad) {
        this.ad = ad;
    }                                   //service层基本不用怎么配置或像编码方式那样写
//    private TransactionTemplate tt;
//    public void setTt(TransactionTemplate tt) {
//        this.tt = tt;
//    }
    @Override
    @Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readOnly=false)
    public void transfer(final Integer from,final Integer to,final Double money) {
        //减钱
        ad.decreaseMoney(from, money);
//        int i = 1/0;
        //加钱
        ad.increaseMoney(to, money);
    }
```

2.2.2 配置文件更改

2.3 注解配置(推荐，最简洁)

2.3.1 service层

```java
@Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readOnly=true)
public class AccountServiceImpl implements AccountService  {
    private AccountDao ad ;

    public void setAd(AccountDao ad) {
        this.ad = ad;
    }
//  private TransactionTemplate tt;
//  public void setTt(TransactionTemplate tt) {
//      this.tt = tt;
//  }
    @Override
    @Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readOnly=false)
    public void transfer(final Integer from,final Integer to,final Double money) {
        //减钱
        ad.decreaseMoney(from, money);
//      int i = 1/0;
        //加钱
        ad.increaseMoney(to, money);
    }
```

//在类上加事务控制，表示对所有方法加同样的事务控制

注解方式

//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

2.3.2 xml配置文件更改

```xml
<context:property-placeholder location="classpath:jdbc.properties"/>

<!-- 事务核心管理器,封装了所有事务操作. 依赖于连接池 -->
<bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
    <property name="dataSource" ref="dataSource" ></property>
</bean>
<!-- 事务模板对象 -->
<!-- <bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate" >
    <property name="transactionManager" ref="transactionManager" ></property>
</bean>
 -->
<!-- 开启使用注解管理aop事务 -->
<tx:annotation-driven/>

<!-- 0.将连接池放入spring容器 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
    <property name="driverClass" value="${jdbc.driverClass}" ></property>
    <property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
    <property name="user" value="${jdbc.user}" ></property>
    <property name="password" value="${jdbc.password}" ></property>
</bean>

<!-- 1.将AccountDao放入spring容器 -->
<bean name="accountDao" class="cn.scct.dao.AccountDaoImpl" >
    <property name="dataSource" ref="dataSource" ></property>
</bean>

<!-- 2.将AccountService放入spring容器 -->
<bean name="accountService" class="cn.scct.service.AccountServiceImpl" >
    <property name="ad" ref="accountDao" ></property>
    <!-- <property name="tt" ref="transactionTemplate" ></property> -->
</bean>
```

还有事务核心管理器，无论如何不能丢

只需要添加这句话