一、DbUtils的使用

DbUtils是Apache的一款用于简化Dao代码的工具类，它底层封装了JDBC技术。

核心类：

QueryRunner 用于执行增删改查的SQL语句

ResultSetHandler 这是一个接口，主要作用是将数据库返回的记录封装进实体对象

核心方法：

update() 用来执行增、删、改语句

query() 用来执行查询语句

举个例子：

```
 //创建一个QueryRunner对象，用来执行增删改查
 //这里需要给一个数据源，如果此处不给，那么使用它调用具体API的时候必须要给
 QueryRunner queryRunner = new QueryRunner(dataSource);

 //调用update方法，执行insert语句
 queryRunner.update("insert into account value(null,?,?)",1,2);

//调用query方法，执行查询语句
//BeanHandler是ResultSetHandler的一个实现类，用于将一条返回数据封装成一个JavaBean
//类似的子类还有BeanListHandler、MapHandler等等
queryRunner.query("select * from account where aid = ?",
                  new BeanHandler<Account>(Account.class),1);
```

二、基于Spring的xml配置实现账户的CRUD案例

2.1 数据库环境准备

```
create table account(
    id int primary key auto_increment,
    name varchar(40),
    money float
)character set utf8 collate utf8_general_ci;

insert into account(name,money) values('aaa',1000);
insert into account(name,money) values('bbb',1000);
insert into account(name,money) values('ccc',1000);
```

2.2 创建工程并导入坐标

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.46</version>
    </dependency>
    <dependency>
        <groupId>commons-dbutils</groupId>
        <artifactId>commons-dbutils</artifactId>
        <version>1.6</version>
    </dependency>
    <dependency>
        <groupId>c3p0</groupId>
        <artifactId>c3p0</artifactId>
        <version>0.9.1.2</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

2.3 实体类

```java
public class Account {
    private Integer id;
    private String name;
    private Double money;
}
```

2.4 编写持久层代码

```java
public interface AccountDao {

    //保存
    void save(Account account);

    //根据主键查询
    Account findByAid(Integer aid);

    //查询所有
    List<Account> findAll();
```

```
    //根据账户名称修改金额
    void update(Account account);

    //根据主键删除
    void deleteByAid(Integer aid);

}
```

```
public class AccountDaoImpl implements AccountDao {

    private QueryRunner queryRunner;
    public void setQueryRunner(QueryRunner queryRunner) {
        this.queryRunner = queryRunner;
    }

    @Override
    public Account findById(Integer id) {
        Account account = null;
        try {
            account = queryRunner.query("select * from account where id = ?", new
BeanHandler<Account>(Account.class), id);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return account;
    }

    @Override
    public List<Account> findAll() {
        List<Account> list = new ArrayList<>();
        try {
            list = queryRunner.query("select * from account", new BeanListHandler<Account>
(Account.class));
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return list;
    }

    @Override
    public void save(Account account) {
        try {
            queryRunner.update("insert into account (name, money) values (?, ?)",
account.getName(), account.getMoney());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void update(Account account) {

        try {
```

```
            queryRunner.update("update account set name=?, money=? where id=?",
                    account.getName(), account.getMoney(), account.getId());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void delete(Integer id) {
        try {
            queryRunner.update("delete from account where id=?", id);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

2.5 编写业务层代码

```
public interface AccountService {

    public Account findById(Integer id);

    public List<Account> findAll();

    public void save(Account account);

    public void update(Account account);

    public void delete(Integer id);
}
```

```
public class AccountServiceImpl implements AccountService {

    private AccountDao accountDao;
    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    @Override
    public Account findById(Integer id) {
        return accountDao.findById(id);
    }

    @Override
    public List<Account> findAll() {
        return accountDao.findAll();
    }
```

```
    @Override
    public void save(Account account) {
        accountDao.save(account);
    }

    @Override
    public void update(Account account) {
        accountDao.update(account);
    }

    @Override
    public void delete(Integer id) {
        accountDao.delete(id);
    }
}
```

2.6 创建spring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"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context.xsd
                http://www.springframework.org/schema/aop
                http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!--把数据库连接池对象交给IOC容器-->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql:///spring"/>
        <property name="user" value="root"/>
        <property name="password" value="root"/>
    </bean>
    <!--把QueryRunner对象交给IOC容器-->
    <bean id="queryRunner" class="org.apache.commons.dbutils.QueryRunner">
        <constructor-arg name="ds" ref="dataSource"/>
    </bean>
    <!--把dao对象交给IOC容器-->
    <bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl">
        <property name="queryRunner" ref="queryRunner"/>
    </bean>
    <!--把service对象交给IOC容器-->
    <bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl">
        <property name="accountDao" ref="accountDao"/>
    </bean>

</beans>
```

2.7 测试

```
package com.itheima.test;

import com.itheima.domain.Account;
import com.itheima.service.AccountService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.List;

public class AccountTest {

    @Test
    public void findById(){
        //加载IOC容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
        AccountService accountService = ac.getBean(AccountService.class);
        Account account = accountService.findById(1);
        System.out.println(account.getName());
    }

    @Test
    public void findAll(){
        //加载IOC容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
        AccountService accountService = ac.getBean(AccountService.class);
        List<Account> list = accountService.findAll();
        for (Account account : list) {
            System.out.println(account.getName());
        }
    }

    @Test
    public void save(){
        //加载IOC容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
        AccountService accountService = ac.getBean(AccountService.class);
        Account account = new Account();
        account.setName("ddd");
        account.setMoney(1000d);
        accountService.save(account);
    }

    @Test
    public void update(){
        //加载IOC容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
        AccountService accountService = ac.getBean(AccountService.class);
        Account account = new Account();

        account.setId(11);
```

```
        account.setName("ddd");
        account.setMoney(100d);
        accountService.update(account);
    }


    @Test
    public void delete(){
        //加载IOC容器
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
        AccountService accountService = ac.getBean(AccountService.class);
        accountService.delete(11);
    }
}
```

三、spring中常用注解介绍

学习基于注解的 IoC 配置，大家脑海里首先得有一个认知，即注解配置和 xml 配置要实现的功能都是一样的，都是要降低程序间的耦合。只是配置的形式不一样。

3.1 注解开发环境说明

3.1.1 导入jar包

```
<!--spring5版本注解开发除了四大核心包，还需要AOP。而context包本来就依赖于AOP包。-->
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>
</dependencies>
```

3.1.2 开启组件扫描

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

```
    <!--
        组件扫描中base-package指定一个包名，
        spring会自动扫描当前包及其子包中的类，
        如果扫描到类上有IOC注解，就会把当前类交给IOC容器管理。
        如果扫描到属性上有DI注解，则依据依赖注入的规则，给属性注入值。
     -->
    <context:component-scan base-package="com.itheima"/>

</beans>
```

3.2 IOC注解说明

IOC注解都必须放在想要被IOC容器管理的类上，一共有四个：

@Component 非三层结构范围的类上使用 @Controller 一般标注在变现层的类上 @Service 一般标注在业务层的类上 @Repository 一般标注在持久层的类上

3.3 bean对象作用域注解说明

@Scope

用于指定bean的作用范围，相当于配置文件中的< bean scope="">

3.4 bean对象创建后和摧毁前触发行为的注解

@PostConstruct bean对象创建后触发行为

@PreDestroy bean对象摧毁前触发行为

相当于< bean init-method="init" destroy-method="destory" />

可用于测试bean对象生命周期

3.5 DI注解说明

DI注解都相当于直接给属性赋值，而无需借助于set方法或构造方法。

@Autowired

放在属性上：

表示先按照类型给属性注入值【by type】

如果IOC容器中存在多个与属性同类型的对象，则会按照属性名注入值【by name】

也可以配合@Qualifier("IOC容器中对象id")注解直接按照名称注入值

放在方法上：

表示自动执行当前方法，如果方法有参数，会自动从IOC容器中寻找同类型的对象给参数传值

也可以在参数上添加@Qualifier("IOC容器中对象id")注解按照名称寻找对象给参数传值

@Resource

只能放在属性上，表示先按照属性名匹配IOC容器中对象id给属性注入值【by name】

若没有成功，会继续根据当前属性的类型匹配IOC容器中同类型对象来注入值 【by type】

若指定了name属性@Resource(name = "对象id")，则只能按照对象id注入值

@Value

用于简单数据类型的注入，相当于< property name="" value="" >,但通常不这么使用

此注解一般用于解析其它properties配置文件中的key值俩获取对应的value，写法如下：

@Value("${key}")

3.6 总结

- 注解和xml是Spring提供的两种配置形式，所实现的功能完全一样。
- 注解的好处是配置简单，xml的好处是修改配置不用改动源码，企业开发中两种方式灵活使用。
- 注意在注解使用前不要忘记添加包扫描< context:component-scan base-package="" />
- 注解和xml配置对应关系如下表格:

xml配置	注解配置	说明
< bean id="" class="" >	@Component @Controller @Service @Repository	bean的实例化
< property name="" ref="">	@Autowired、@Qualifier、@Resource	bean的对象属性注入
< property name="" value="">	@Value	bean的简单属性注入
< bean scope="">	@Scope	控制bean的作用范围
< bean init-method="init" destroy-method="destory" />	@PostConstruct @PreDestroy	bean的创建后和销毁前调用的方法

四、Spring常用注解实现账户的CRUD案例

注：拷贝纯xml版的案例工程进行修改

4.1 修改spring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"

        xmlns:context="http://www.springframework.org/schema/context"
```

```
        xmlns:aop="http://www.springframework.org/schema/aop"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context.xsd
                http://www.springframework.org/schema/aop
                http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!--组件扫描-->
    <context:component-scan base-package="com.itheima"/>
    <!--把数据库连接池对象交给IOC容器-->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql:///spring"/>
        <property name="user" value="root"/>
        <property name="password" value="root"/>
    </bean>
    <!--把QueryRunner对象交给IOC容器-->
    <bean id="queryRunner" class="org.apache.commons.dbutils.QueryRunner">
        <constructor-arg name="ds" ref="dataSource"/>
    </bean>

</beans>
```

4.2 修改dao实现类

```
@Repository
public class AccountDaoImpl implements AccountDao {

    @Autowired
    private QueryRunner queryRunner;

    //其余代码跟原来一模一样
}
```

4.3 修改service实现类

```
@Service
public class AccountServiceImpl implements AccountService {

    @Autowired
    private AccountDao accountDao;

    //其余代码跟原来一模一样

}
```

4.4 测试

```
//依旧使用原来的测试代码即可
```

五、 spring新注解说明

5.1 @Configuration

用于指定当前类是一个 spring配置类，当创建容器时会从该类上加载注解 。

```
//声明这是一个配置类，Spring会像从配置文件读取内容似的，从这个类中读取内容
@Configuration
public class SpringConfiguration {

}
```

5.2 @Bean

该注解只能写在方法上，表明使用此方法创建一个对象，并且放入 spring 容器。它支持一个name属性，用于给生成的bean取一个id。

5.3 @ComponentScan

组件扫描注解。 相当于xml配置文件中的< context:component-scan base-package=""/>

5.4 @PropertySource

用于加载.properties 文件中的配置。例如我们配置数据源时，可以把连接数据库的信息写到properties 配置文件中，就可以使用此注解指定 properties 配置文件的位置。

5.5 @Import

在一个配置类中，引入其他配置，具体写法：@Import(DaoConfig.class)

效果等于之前xml配置中的

六、spring纯注解实现账户的CRUD案例【了解】

注：复制spring常用注解案例做修改

6.1 直接删掉applicationContext.xml配置文件

略

6.2 提供jdbc.properties配置文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///spring
jdbc.username=root
jdbc.password=root
```

6.3 提供数据库操作相关信息配置类

```
@Configuration
@PropertySource("jdbc.properties")
public class DaoConfig {
    @Value("${jdbc.driver}")
    private String driver;
    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.username}")
    private String username;
    @Value("${jdbc.password}")
    private String password;

    @Bean("dataSource")
    public DataSource createDataSource(){
        try {
            ComboPooledDataSource dataSource = new ComboPooledDataSource();
            dataSource.setDriverClass(driver);
            dataSource.setJdbcUrl(url);
            dataSource.setUser(username);
            dataSource.setPassword(password);
            return dataSource;
        }catch (Exception e){
            throw new RuntimeException("创建数据库连接池失败！");
        }
    }

    @Bean
    public QueryRunner createQueryRunner(@Qualifier("dataSource") DataSource dataSource){
        return new QueryRunner(dataSource);
    }

}
```

6.4 提供spring主配置类

```
@ComponentScan("com.itheima")
@Import(DaoConfig.class)
public class SpringConfig {

}
```

6.5 测试

```
//注意这时加载的是配置类，而不是配置文件
@Test
public void findById(){
    //加载IOC容器
    ApplicationContext ac = new AnnotationConfigApplicationContext(SpringConfig.class);
    AccountService accountService = ac.getBean(AccountService.class);
    Account account = accountService.findById(1);
    System.out.println(account.getName());
}
```

七、Spring整合Junit

这是Spring提供的对Junit单元测试的一种支持。使用步骤如下:

1）引入坐标

```
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>5.1.6.RELEASE</version>
</dependency>
```

2）在测试类上使用@RunWith指定Spring的单元测试运行器

```
@RunWith(SpringJUnit4ClassRunner.class)
public class AccountTest {
}
```

3）使用@ContextConfiguration指定配置文件，它支持文件和类的形式

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = SpringConfig.class)
public class AccountTest {
}
```

4）下面可以直接使用@Autowired的形式，让Spring容器为我们注入需要的bean了。

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = SpringConfig.class)
public class AccountTest {
    @Autowired
    private AccountService accountService;
}
```

八、作业

1、完成今天的代码

要求：两个版本

第一个版本： xml 和注解组合使用

第二个版本：纯注解配置

2、在业务层添加一个转账方法

实现账户名称 aaa 给账户名称 bbb 转账功能。

在转账方法中模拟转账异常，并分析问题产生的原因。

解决转账异常带来的事务问题（不满足事务的一致性）