# Spring框架第二天 -- 拓薪教育

## 第一章：Spring框架开发程序的方式

### 1. Spring框架开发方式

1. 需求：编写service和dao的类，演示代码
2. 技术选择：持久层使用原始的JDBC的程序，连接池选择的是Druid连接池。创建maven工程，导入开发的jar包

<dependencies>  
        <dependency>  
            <groupId>org.springframework</groupId>  
            <artifactId>spring-context</artifactId>  
            <version>5.0.2.RELEASE</version>  
        </dependency>  
        <dependency>  
            <groupId>commons-logging</groupId>  
            <artifactId>commons-logging</artifactId>  
            <version>1.2</version>  
        </dependency>  
        <dependency>  
            <groupId>log4j</groupId>  
            <artifactId>log4j</artifactId>  
            <version>1.2.12</version>  
        </dependency>  
        <dependency>  
            <groupId>junit</groupId>  
            <artifactId>junit</artifactId>  
            <version>4.12</version>  
            <scope>test</scope>  
        </dependency>  
        <!--连接池-->  
        <dependency>  
            <groupId>com.alibaba</groupId>  
            <artifactId>druid</artifactId>  
            <version>1.1.10</version>  
        </dependency>  
        <!--mysql驱动包-->  
        <dependency>  
            <groupId>mysql</groupId>  
            <artifactId>mysql-connector-java</artifactId>  
            <version>5.1.6</version>  
        </dependency>  
   
    </dependencies>

创建数据库，创建表结构

create database spring\_db;  
use spring\_db;  
create table account(  
 id int primary key auto\_increment,  
 name varchar(40),  
 money double  
)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);

编写JavaBean的类

package cn.tx.domain;  
​  
import java.io.Serializable;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Account implements Serializable {  
​  
    private static final long serialVersionUID = 7355810572012650248L;  
​  
    private Integer id;  
    private String name;  
    private Double 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 Double getMoney() {  
        return money;  
   }  
    public void setMoney(Double money) {  
        this.money = money;  
   }  
​  
    @Override  
    public String toString() {  
        return "Account{" +  
                "id=" + id +  
                ", name='" + name + '\'' +  
                ", money=" + money +  
                '}';  
   }  
}  
​

编写AccountDao的接口和实现类

package cn.tx.dao;  
​  
import cn.tx.domain.Account;  
​  
import java.util.List;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public interface AccountDao {  
​  
    public List<Account> findAll();  
​  
}  
   
 package cn.tx.dao;  
​  
   import cn.tx.domain.Account;  
​  
   import com.alibaba.druid.pool.DruidDataSource;  
​  
   import javax.sql.DataSource;  
​  
   import java.sql.Connection;  
​  
   import java.sql.PreparedStatement;  
​  
   import java.sql.ResultSet;  
​  
   import java.sql.SQLException;  
​  
   import java.util.ArrayList;  
​  
   import java.util.List;  
​  
    
/\*\*  
\* 拓薪教育 -- 腾讯课程认证机构  
\* 樱木老师  
\*/  
public class AccountDaoImpl implements AccountDao {    
 // 注入连接池对象  
   private DataSource dataSource;  
   public void setDataSource(DataSource dataSource) {  
       this.dataSource = dataSource;  
   }  
​  
   /\*\*  
    \* 查询所有的数据  
    \* @return  
    \*/  
   @Override  
   public List<Account> findAll() {  
​  
       /\*  
       DruidDataSource dataSource = new DruidDataSource();  
       dataSource.setDriverClassName("com.mysql.jdbc.Driver");  
       dataSource.setUrl("jdbc:mysql:///spring\_db");  
       dataSource.setUsername("root");  
       dataSource.setPassword("root");  
       \*/  
​  
       List<Account> list = new ArrayList<>();  
​  
       Connection connection = null;  
       PreparedStatement stmt = null;  
       ResultSet rs = null;  
​  
       try {  
           // 获取连接  
           connection = dataSource.getConnection();  
           // 编写sql语句  
           String sql = "select \* from account";  
           // 预编译  
           stmt = connection.prepareStatement(sql);  
           // 查询  
           rs = stmt.executeQuery();  
           // 遍历，封装数据  
           while (rs.next()){  
               Account account = new Account();  
               account.setId(rs.getInt("id"));  
               account.setName(rs.getString("name"));  
               account.setMoney(rs.getDouble("money"));  
               list.add(account);  
           }  
       } catch (SQLException e) {  
           e.printStackTrace();  
       }finally {  
           try {  
               connection.close();  
           } catch (SQLException e) {  
               e.printStackTrace();  
           }  
           try {  
               stmt.close();  
           } catch (SQLException e) {  
               e.printStackTrace();  
           }  
           try {  
               rs.close();  
           } catch (SQLException e) {  
               e.printStackTrace();  
           }  
       }  
       return list;  
   }   
 }  
​

​  
   package cn.tx.service;  
​  
   import cn.tx.domain.Account;  
​  
   import java.util.List;  
​  
   /\*\*  
    \* 拓薪教育 -- 腾讯课程认证机构  
    \* 樱木老师  
    \*/  
   public interface AccountService {  
​  
       public List<Account> findAll();  
​  
   }  
​  
​  
   package cn.tx.service;  
​  
   import cn.tx.dao.AccountDao;  
   import cn.tx.domain.Account;  
​  
   import java.util.List;  
​  
   /\*\*  
    \* 拓薪教育 -- 腾讯课程认证机构  
    \* 樱木老师  
    \*/  
   public class AccountServiceImpl implements AccountService {  
​  
       // 依赖注入  
       private AccountDao accountDao;  
       public void setAccountDao(AccountDao accountDao) {  
           this.accountDao = accountDao;  
       }  
​  
       /\*\*  
        \* 查询所有的数据  
        \* @return  
        \*/  
       @Override  
       public List<Account> findAll() {  
           return accountDao.findAll();  
       }  
​  
   }

编写配置文件

<?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">  
​  
    <!--配置连接池-->  
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">  
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />  
        <property name="url" value="jdbc:mysql:///spring\_db" />  
        <property name="username" value="root" />  
        <property name="password" value="root" />  
    </bean>  
​  
    <!--管理bean-->  
    <bean id="accountService" class="cn.tx.service.AccountServiceImpl">  
        <property name="accountDao" ref="accountDao" />  
    </bean>  
    <bean id="accountDao" class="cn.tx.dao.AccountDaoImpl">  
        <property name="dataSource" ref="dataSource" />  
    </bean>  
​  
</beans>

编程测试程序

package cn.tx.test;  
​  
import cn.tx.domain.Account;  
import cn.tx.service.AccountService;  
import org.junit.Test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
​  
import java.util.List;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Demo1 {  
​  
    @Test  
    public void run1(){  
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");  
        AccountService accountService = (AccountService) ac.getBean("accountService");  
        // 调用方法  
        List<Account> list = accountService.findAll();  
        for (Account account : list) {  
            System.out.println(account);  
       }  
   }  
​  
}

## 第二章：IOC注解的方式

### 1. IOC注解方式的快速入门

IOC注解的方式依赖没有变化

编写接口和实现类

package cn.tx.demo2;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public interface UserService {  
​  
    public void hello();  
​  
}  
​  
package cn.tx.demo2;  
​  
import org.springframework.stereotype.Component;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* <bean id="us" class="cn.tx.demo2.UserServiceImpl" />  
 \*/  
// 组件，作用：把当前类使用IOC容器进行管理，如果没有指定名称，默认使用类名，首字母是小写。userServiceImpl。或者自己指定名称  
@Component(value = "us")  
public class UserServiceImpl implements UserService {  
​  
    @Override  
    public void hello() {  
        System.out.println("Hello IOC注解...");  
   }  
​  
}

在需要管理的类上添加@Component注解

@Component(value = "us")  
public class UserServiceImpl implements UserService {  
​  
    @Override  
    public void hello() {  
        System.out.println("Hello IOC注解...");  
   }  
​  
}

编写配置文件，重点是开启注解扫描。

<?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" 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">  
​  
    <!--开启注解扫描  
    <context:component-scan base-package="cn.tx.demo2" />  
    -->  
​  
    <!--开启注解扫描 cn.tx.所有的包中的所有的类 -->  
    <context:component-scan base-package="cn.tx" />  
​  
</beans>

编写测试方法

package cn.tx.test;  
​  
import cn.tx.demo2.UserService;  
import org.junit.Test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Demo2 {  
​  
    /\*\*  
     \* IOC注解方式的入门  
     \*/  
    @Test  
    public void run1(){  
        // 工厂  
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext\_anno.xml");  
        // 获取对象  
        UserService userService = (UserService) ac.getBean("us");  
        userService.hello();  
   }  
​  
}

### 2. 常用的注解

bean管理类常用的4个注解（作用相同，推荐使用在不同分层上）

​ @Component 普通的类

​ @Controller 表现层

​ @Service 业务层

​ @Repository 持久层

依赖注入常用的注解

​ @Value 用于注入普通类型（String，int，double等类型）

​ @Autowired 默认按类型进行自动装配（引用类型）

​ @Qualifier 和@Autowired一起使用，强制使用名称注入

​ @Resource Java提供的注解，也被支持。使用name属性，按名称注入

对象生命周期（作用范围）注解

​ @Scope 生命周期注解，取值singleton（默认值，单实例）和prototype（多例）

初始化方法和销毁方法注解（了解）

​ @PostConstruct 相当于init-method

​ @PreDestroy 相当于destroy-method

具体的代码如下

package cn.tx.demo3;  
​  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Scope;  
import org.springframework.stereotype.Component;  
import org.springframework.stereotype.Controller;  
import org.springframework.stereotype.Repository;  
import org.springframework.stereotype.Service;  
​  
import javax.annotation.PostConstruct;  
import javax.annotation.Resource;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
// 默认当前类名就是ID名称，首字母小写  
@Component(value = "c")  
// @Controller  
// @Service(value = "c")  
// @Repository(value = "c")  
// @Scope(value = "singleton")     // 默认值，单例的  
// @Scope(value = "prototype")         // 多例的  
public class Car {  
​  
    // 注解注入值，属性set方法是可以省略不写的。  
    // 只有一个属性，属性的名称是value，value是可以省略不写的  
    @Value("大奔2")  
    private String cname;  
​  
    @Value(value = "400000")  
    private Double money;  
​  
    // 也不用提供set方法  
    // 按类型自动装配的注解，和id名称没有关系  
    @Autowired  
    // 按id的名称注入，Qualifier不能单独使用，需要Autowired一起使用。  
    // @Qualifier(value = "person")  
    // @Resource Java提供的注解，按名称注入对象，属性名称是name  
    // @Resource(name = "person")  
    private Person person;  
​  
    /\*\*  
     \* Car对象创建完成后，调用init方法进行初始化操作  
     \*/  
    @PostConstruct  
    public void init(){  
        System.out.println("操作...");  
   }  
​  
    /\*  
    public String getCname() {  
        return cname;  
    }  
​  
    public void setCname(String cname) {  
        this.cname = cname;  
    }  
​  
    public Double getMoney() {  
        return money;  
    }  
​  
    public void setMoney(Double money) {  
        this.money = money;  
    }  
    \*/  
​  
    @Override  
    public String toString() {  
        return "Car{" +  
                "cname='" + cname + '\'' +  
                ", money=" + money +  
                ", person=" + person +  
                '}';  
   }  
}  
​  
​  
package cn.tx.demo3;  
​  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
@Component(value = "person")  
public class Person {  
​  
    @Value("张三")  
    private String pname;  
​  
    @Override  
    public String toString() {  
        return "Person{" +  
                "pname='" + pname + '\'' +  
                '}';  
   }  
​  
}  
​  
​  
package cn.tx.test;  
​  
import cn.tx.demo2.UserService;  
import cn.tx.demo3.Car;  
import org.junit.Test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Demo3 {  
​  
    @Test  
    public void run1(){  
        // 工厂  
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext\_anno.xml");  
        // 获取对象  
        Car car = (Car) ac.getBean("c");  
        System.out.println(car);  
   }  
​  
}

### 3. IOC纯注解的方式

纯注解的方式是微服务架构开发的主要方式，所以也是非常的重要。纯注解的目的是替换掉所有的配置文件。但是需要编写配置类。

编写实体类

package cn.tx.demo4;  
​  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
@Component  
public class Order {  
​  
    @Value("北京")  
    private String address;  
​  
    @Override  
    public String toString() {  
        return "Order{" +  
                "address='" + address + '\'' +  
                '}';  
   }  
​  
}

编写配置类，替换掉applicationContext.xml配置文件

package cn.tx.demo4;  
​  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* Spring的配置类，替换掉applicationContext.xml  
 \*  
 \*/  
// 声明当前类是配置类  
@Configuration  
// 扫描指定的包结构  
@ComponentScan(value = "cn.tx.demo4")  
public class SpringConfig {  
​  
}

测试方法的编写

package cn.tx.test;  
​  
import cn.tx.demo4.Order;  
import cn.tx.demo4.SpringConfig;  
import org.junit.Test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Demo4 {  
​  
    /\*\*  
     \* 编写程序，需要加载配置类  
     \*/  
    @Test  
    public void run1(){  
        // 创建工厂，加载配置类  
        ApplicationContext ac = new AnnotationConfigApplicationContext(SpringConfig.class);  
        // 获取到对象  
        Order order = (Order) ac.getBean("order");  
        System.out.println(order);  
   }  
​  
}

常用的注解总结

@Configuration 声明是配置类

@ComponentScan 扫描具体包结构的

@Import注解 Spring的配置文件可以分成多个配置的，编写多个配置类。用于导入其他配置类

package cn.tx.demo4;  
​  
import org.springframework.context.annotation.Configuration;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* 新的配置类  
 \*  
 \*/  
@Configuration      // 声明配置类  
public class SpringConfig2 {  
​  
}  
​  
​  
// 声明当前类是配置类  
@Configuration  
// 扫描指定的包结构  
@ComponentScan(value = "cn.tx.demo4")  
// @ComponentScan(value = {"cn.tx.demo4","cn.tx.demo3"})  
// 引入新的配置类  
@Import(value = {SpringConfig2.class})  
public class SpringConfig {

@Bean注解 只能写在方法上，表明使用此方法创建一个对象，对象创建完成保存到IOC容器中

package cn.tx.demo4;  
​  
import com.alibaba.druid.pool.DruidDataSource;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Import;  
​  
import javax.sql.DataSource;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* Spring的配置类，替换掉applicationContext.xml  
 \*  
 \*/  
// 声明当前类是配置类  
@Configuration  
// 扫描指定的包结构  
@ComponentScan(value = "cn.tx.demo4")  
// @ComponentScan(value = {"cn.tx.demo4","cn.tx.demo3"})  
// 引入新的配置类  
@Import(value = {SpringConfig2.class})  
public class SpringConfig {  
​  
    /\*\*  
     \* 创建连接池对象，返回对象，把该方法创建后的对象存入到连接池中，使用@Bean注解解决  
​  
         <!--配置连接池对象-->  
         <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">  
             <property name="driverClassName" value="com.mysql.jdbc.Driver" />  
             <property name="url" value="jdbc:mysql:///spring\_db" />  
             <property name="username" value="root" />  
             <property name="password" value="root" />  
         </bean>  
     \*  
     \* @return  
     \*/  
    @Bean(name="dataSource")  
    public DataSource createDataSource(){  
        DruidDataSource dataSource = new DruidDataSource();  
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");  
        dataSource.setUrl("jdbc:mysql:///spring\_db");  
        dataSource.setUsername("root");  
        dataSource.setPassword("root");  
        return dataSource;  
   }  
​  
}  
​

## 第三章：Spring框架整合JUnit单元测试

### 1. Spring框架整合JUnit单元测试

每次进行单元测试的时候，都需要编写创建工厂，加载配置文件等代码，比较繁琐。Spring提供了整合Junit单元测试的技术，可以简化测试开发。下面开始学习该技术。

必须先有Junit单元测试的环境，也就是说已经导入Junit单元测试的jar包。咱们已经导入过了。使用的是4.12版本

再导入spring-test的坐标依赖

<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-test</artifactId>  
    <version>5.0.2.RELEASE</version>  
    <scope>test</scope>  
</dependency>

编写类和方法，把该类交给IOC容器进行管理

package cn.tx.demo5;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class User {  
​  
    public void sayHello(){  
        System.out.println("Hello....");  
   }  
​  
}

编写配置文件applicationContext\_test.xml

<?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">  
​  
    <!--整合单元测试-->  
    <bean id="user" class="cn.tx.demo5.User"/>  
​  
</beans>

编写测试代码

package cn.tx.test;  
​  
import cn.tx.demo5.User;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.test.context.ContextConfiguration;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* Spring整合Junit单元测试  
 \*  
 \*/  
@RunWith(value = SpringJUnit4ClassRunner.class)     // 运行单元测试  
@ContextConfiguration(value = "classpath:applicationContext\_test.xml")   // 加载类路径下的配置文件  
public class Demo5 {  
​  
    // 测试哪一个对象，把该对象注入进来，在测试环境下，可以使用注解的方式注入测试的对象  
    // 按类型自动注入  
    @Autowired  
    private User user;  
​  
    @Test  
    public void run1(){  
        // 创建工厂，加载配置文件......  
        // 调用对象的方法  
        user.sayHello();  
   }  
​  
}

### 2. Spring整合单元测试（纯注解方式）

编写类和方法

package cn.tx.demo6;  
​  
import org.springframework.stereotype.Component;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
@Component  
public class Customer {  
   
    public void save(){  
        System.out.println("保存客户...");  
   }  
​  
}

编写配置类

package cn.tx.demo6;  
​  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* Spring整合Junit配置类  
 \*/  
// 声明  
@Configuration  
// 扫描包结构  
@ComponentScan(value = "cn.tx.demo6")  
public class SpringConfig6 {  
}

编写测试方法

package cn.tx.test;  
​  
import cn.tx.demo6.Customer;  
import cn.tx.demo6.SpringConfig6;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.test.context.ContextConfiguration;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* Spring整合Junit 注解的方式测试  
 \*/  
@RunWith(SpringJUnit4ClassRunner.class)  
// 加载配置类  
@ContextConfiguration(classes = SpringConfig6.class)  
public class Demo6 {  
​  
    // 按类型注入  
    @Autowired  
    private Customer customer;  
​  
    /\*\*  
     \* 测试  
     \*/  
    @Test  
    public void run1(){  
        customer.save();  
   }  
​  
}  
​