# Spring框架第一天 -- 拓薪教育

## 第一章：Spring框架的介绍

### 1. Spring框架的概述

Spring是一个开放源代码的设计层面框架，它解决的是业务逻辑层和其他各层的松耦合问题，因此它将面向接口的编程思想贯穿整个系统应用。Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson创建。简单来说，Spring是一个分层的JavaSE/EE full-stack(一站式) 轻量级开源框架。

Spring是于2003 年兴起的一个轻量级的Java开发框架，由Rod Johnson在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。

它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许使用者选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。

Spring的核心是控制反转（IoC）和面向切面（AOP）。简单来说，Spring是一个分层的JavaSE/EEfull-stack(一站式) 轻量级开源框架。

### 2. Spring框架的优点

方便解耦，简化开发，Spring就是一个大工厂，可以将所有对象创建和依赖关系维护，交给Spring管理。IOC的作用。

AOP编程的支持，Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能。

声明式事务的支持，只需要通过配置就可以完成对事务的管理，而无需手动编程。

方便程序的测试，Spring对Junit4支持，可以通过注解方便的测试Spring程序。

方便集成各种优秀框架，Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts2、Hibernate、MyBatis、Quartz等）的直接支持。

降低JavaEE API的使用难度，Spring 对JavaEE开发中非常难用的一些API（JDBC、JavaMail、远程调用等），都提供了封装，使这些API应用难度大大降低。

## 第二章：Spring的IOC核心技术

### 1. 什么是IOC

IOC -- Inverse of Control，控制反转，将对象的创建权力反转给Spring框架！！

控制反转（Inversion of Control，缩写为IoC），是面向对象编程中的一种设计原则，可以用来减低计算机代码之间的耦合度。

解决问题：使用IOC可以解决的程序耦合性高的问题！！Spring的工厂读取配置文件。

### 2. IOC的程序入门

创建Java工程，导入坐标依赖

<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>  
    </dependencies>

编写接口和实现类，编写具体的实现方法

package cn.tx.service;  
​  
/\*\*  
 \* 拓薪教育：樱木老师  
 \*/  
public interface UserService {  
​  
    public void hello();  
​  
}  
​  
​  
package cn.tx.service;  
​  
/\*\*  
 \* 拓薪教育：樱木老师  
 \*/  
public class UserServiceImpl implements UserService {  
    @Override  
    public void hello() {  
        System.out.println("Hello IOC!!");  
   }  
}

编写Spring核心的配置文件，在src目录下创建applicationContext.xml的配置文件，名称是可以任意的，但是一般都会使用默认名称。

<?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">  
      
    <!--IOC管理bean-->  
    <bean id="userService" class="cn.tx.service.UserServiceImpl" />  
   
</beans>

把log4j.properties的配置文件拷贝到resources目录下，做为log4j的日志配置文件。

编写测试方法。

package cn.tx.test;  
​  
import cn.tx.service.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 Demo1 {  
​  
    /\*\*  
     \* 入门程序  
     \*/  
    @Test  
    public void run1(){  
        // 使用Spring的工厂  
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");  
        // 通过工厂获得类:  
        UserService userService = (UserService) applicationContext.getBean("userService");  
        userService.hello();  
   }  
​  
}

### 3. IOC技术总结

ApplicationContext接口，工厂的接口，使用该接口可以获取到具体的Bean对象。该接口下有两个具体的实现类。

ClassPathXmlApplicationContext，加载类路径下的Spring配置文件。

FileSystemXmlApplicationContext，加载本地磁盘下的Spring配置文件。

### 4. Spring框架的Bean管理的配置文件方式

id属性，Bean起个名字，在约束中采用ID的约束，唯一，取值要求：必须以字母开始，可以使用字母、数字、连字符、下划线、句话、冒号 id:不能出现特殊字符。

class属性，Bean对象的全路径。

scope属性，scope属性代表Bean的作用范围。

​ singleton单例（默认值），最常用的方式。

​ prototype多例

​ request应用在Web项目中,每次HTTP请求都会创建一个新的Bean

​ session应用在Web项目中,同一个HTTP Session 共享一个Bean

Bean对象的创建和销毁的两个属性配置

说明：Spring初始化bean或销毁bean时，有时需要作一些处理工作，因此spring可以在创建和拆卸bean的时候调用bean的两个生命周期方法

init-method，当bean被载入到容器的时候调用init-method属性指定的方法

destroy-method，当bean从容器中删除的时候调用destroy-method属性指定的方法

### 5. 实例化Bean对象的三种方式

默认是无参数的构造方法（默认方式，基本上使用）

<bean id="us" class="cn.tx.service.UserServiceImpl" />

静态工厂实例化方式

package cn.tx.demo1;  
​  
import cn.tx.service.UserService;  
import cn.tx.service.UserServiceImpl;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \* 静态工厂方式  
 \*/  
public class StaticFactory {  
​  
    // 静态工厂方式  
    public static UserService createUs(){  
        System.out.println("通过静态工厂的方式创建UserServiceImpl对象...");  
        // 编写很多业务逻辑 权限校验  
        return new UserServiceImpl();  
   }  
​  
}  
​  
<bean id="us" class="cn.tx.demo1.StaticFactory" factory-method="createUs" />

实例工厂实例化方式

package cn.tx.demo1;  
​  
import cn.tx.service.UserService;  
import cn.tx.service.UserServiceImpl;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* 动态工厂方式  
 \*  
 \*/  
public class Dfactory {  
​  
    public UserService createUs(){  
        System.out.println("实例化工厂的方式...");  
        return new UserServiceImpl();  
   }  
​  
}  
​  
<bean id="dfactory" class="cn.tx.demo1.Dfactory" />  
<bean id="us" factory-bean="dfactory" factory-method="createUs" />

## 第三章：DI依赖注入

### 1. 依赖注入的概述

IOC和DI的概念

​ IOC：Inverse of Control，控制反转，将对象的创建权反转给Spring！！

​ DI：Dependency Injection，依赖注入，在Spring框架负责创建Bean对象时，动态的将依赖对象注入到Bean组件中！！

### 2. 属性的set方法注入值

编写属性，提供该属性对应的set方法，编写配置文件完成属性值的注入

package cn.tx.service;  
​  
import cn.tx.dao.OrderDao;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class OrderServiceImpl implements OrderService {  
​  
    // 编写成员属性，一定需要提供该属性的set方法  
    private OrderDao orderDao;  
    // 一定需要提供该属性的set方法，IOC容器底层就通过属性的set方法方式注入值  
    public void setOrderDao(OrderDao orderDao) {  
        this.orderDao = orderDao;  
   }  
​  
    // 消息  
    private String msg;  
    // 年龄  
    private int age;  
​  
    public void setMsg(String msg) {  
        this.msg = msg;  
   }  
​  
    public void setAge(int age) {  
        this.age = age;  
   }  
​  
    @Override  
    public void saveOrder() {  
        System.out.println("业务层：保存订单..."+msg+" - "+age);  
        // 调用  
        orderDao.saveOrder();  
   }  
​  
}  
​  
​  
package cn.tx.dao;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class OrderDaoImpl implements OrderDao {  
​  
    @Override  
    public void saveOrder() {  
        System.out.println("持久层：保存订单...");  
   }  
​  
}  
​  
​  
    <!--DI：依赖注入-->  
    <bean id="os" class="cn.tx.service.OrderServiceImpl">  
        <property name="orderDao" ref="od" />  
        <property name="msg" value="你好" />  
        <property name="age" value="30" />  
    </bean>  
    <bean id="od" class="cn.tx.dao.OrderDaoImpl"></bean>

### 3. 属性构造方法方式注入值

对于类成员变量，构造函数注入。

package cn.tx.demo2;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Car {  
​  
    // 名称  
    private String cname;  
    // 金额  
    private Double money;  
​  
    public Car(String cname, Double money) {  
        this.cname = cname;  
        this.money = money;  
   }  
​  
    @Override  
    public String toString() {  
        return "Car{" +  
                "cname='" + cname + '\'' +  
                ", money=" + money +  
                '}';  
   }  
​  
}  
​  
<bean id="car" class="cn.tx.demo2.Car">  
    <constructor-arg name="cname" value="大奔" />  
 <constructor-arg name="money" value="400000" />  
 </bean>

### 4. 数组，集合(List,Set,Map)，Properties等的注入

package cn.tx.demo3;  
​  
import java.util.Arrays;  
import java.util.List;  
import java.util.Map;  
import java.util.Properties;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class CollectionBean {  
​  
    // 数组  
    private String [] strs;  
    public void setStrs(String[] strs) {  
        this.strs = strs;  
   }  
​  
    private List<String> list;  
    public void setList(List<String> list) {  
        this.list = list;  
   }  
​  
    private Map<String,String> map;  
    public void setMap(Map<String, String> map) {  
        this.map = map;  
   }  
​  
    private Properties properties;  
    public void setProperties(Properties properties) {  
        this.properties = properties;  
   }  
​  
    @Override  
    public String toString() {  
        return "CollectionBean{" +  
                "strs=" + Arrays.toString(strs) +  
                ", list=" + list +  
                ", map=" + map +  
                ", properties=" + properties +  
                '}';  
   }  
}  
​  
    <!--给集合属性注入值-->  
    <bean id="collectionBean" class="cn.tx.demo3.CollectionBean">  
        <property name="strs">  
            <array>  
                <value>美美</value>  
                <value>小凤</value>  
            </array>  
        </property>  
        <property name="list">  
            <list>  
                <value>熊大</value>  
                <value>熊二</value>  
            </list>  
        </property>  
        <property name="map">  
            <map>  
                <entry key="aaa" value="老王"/>  
                <entry key="bbb" value="小王"/>  
            </map>  
        </property>  
        <property name="properties">  
            <props>  
                <prop key="username">root</prop>  
                <prop key="password">123456</prop>  
            </props>  
        </property>  
    </bean>

## 第四章：多配置文件方式

### 1. 多配置文件的加载方式

在src的目录下又多创建了一个配置文件，现在是两个核心的配置文件，那么加载这两个配置文件的方式有两种！

主配置文件中包含其他的配置文件:  
<import resource="applicationContext2.xml"/>  
​  
工厂创建的时候直接加载多个配置文件:  
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(  
"applicationContext.xml","applicationContext2.xml");

​