## 什么是Spring?

Spring 就是一个容器能够管理各层的Bean,并且可以配置bean和bean之间的关系。

在Java代码如果想要得到一个bean可以通过ApplicationContext的getBean获得，spring是依赖反射机制的，即（dom4j + 反射），所以在Spring配置文件中bean 的id属性和class属性中要写全路径。

## 什么是IOC?

IOC（inverse of control）控制反转：所谓控制反转就是把对象（bean）对象和维护对象（bean）之间的关系的权利转移到Sqring容器中去了（ApplicationContext.xml）而程序本身不再维护。

IoC Inverse of Control 反转控制的概念，就是将原本在程序中手动创建对象的控制权，交由Spring框架管理，简单说，就是创建对象控制权被反转到了Spring框架

## 什么是DI?

DI（dependency injection）依赖注入：实际上DI和IOC是同一个概念，因为在ApplicationContext.xml配置文件中bean和bean之间通过ref来维护的时候是相互依赖的，所以又叫做依赖注入。也就是控制反转。

因为ApplicationContext是非常消耗内存的，所以必须保证一个项目里只有一个ApplicationContext实例：

那么如何保证这有一个实例呢，就需要把ApplicationContext对象做成单例形式

## Spring框架Bean实例化的方式

1. 构造方法实例化（默认无参数）
2. 静态工厂实例化
3. 实例工厂实例化

静态工厂实例化：就是在一个工厂类中添加一个静态方法来返回一个return new BeanName();在配置文件中class需填写工厂类，然后添加Factory-method 方法

实例工厂实例化： <!-- 第三种使用实例工厂实例化 -->

<bean id="bean3" factory-bean="bean3Factory" factory-method="getBean3"></bean>

<bean id="bean3Factory" class="cn.itcast.spring3.demo2.Bean3Factory"/>

## Spring中Bean的生命周期

1、instantiate bean对象实例化

2、populate properties 封装属性

3、如果Bean实现BeanNameAware 执行 setBeanName

4、如果Bean实现BeanFactoryAware 或者 ApplicationContextAware 设置工厂

setBeanFactory 或者上下文对象 setApplicationContext

5、如果存在类实现 BeanPostProcessor（后处理Bean） ，执行postProcessBeforeInitialization

6、如果Bean实现InitializingBean 执行 afterPropertiesSet

7、调用<bean init-method="init"> 指定初始化方法 init

8、如果存在类实现 BeanPostProcessor（处理Bean） ，执行postProcessAfterInitialization

9、执行业务处理

10、如果Bean实现 DisposableBean 执行 destroy

11、调用<bean destroy-method="customerDestroy"> 指定销毁方法 customerDestroy

注意：这里的后处理Bean可以帮助我们实现很多功能，例如权限校验等

第五步中是初始化之前，没法找到bean中的方法，所以只能够在第八步中去实现增强的方法

## Bean中的属性注入

1. set方法注入
2. 构造器方法注入
3. 接口注入

Spring目前支持前两种

SpEl 支持属性的注入 #{ } 之前使用ref来引用注入另一个变量，现在可以只需要使用#{ BeanId}

集合类属性的注入：

List(数组)，set，map





## 使用多个XML文件

* 方式一 可以在创建ApplicationContext对象时传入多个配置文件

ApplicationContext applicationContext = new

ClassPathXmlApplicationContext("beans1.xml", "beans2.xml");

* 方式二 可以在配置文件中通过<import>引入其他配置文件

<import resource="classpath:bean2.xml"/>

## 注解



使用注解定义Bean

* 除了@Component外，Spring提供了3个功能基本和@Component等效的注解
  + @Repository 用于对DAO实现类进行标注
  + @Service 用于对Service实现类进行标注
  + @Controller 用于对Controller实现类进行标注

### 1、<context:annotation-config/>和<context:component-scan>之间的区别

<context:annotation-config/>的作用是向spring容器中注入属性（就是那些提供set方法的）

AutowiredAnnotationBeanPostProcessor、CommonAnnotationBeanPostProcessor、PersistenceAnnotationBeanPostProcessor 以及 RequiredAnnotationBeanPostProcessor 这 4 个BeanPostProcessor。注册这4个BeanPostProcessor的作用，就是为了你的系统能够识别相应的注解。

具体来说：如果你想使用@Autowired注解，那么就必须事先在 Spring 容器中声明 AutowiredAnnotationBeanPostProcessor Bean。声明方式如下：<bean class="org.springframework.beans.factory.annotation. AutowiredAnnotationBeanPostProcessor "/>如果想使用@ Resource 、@ PostConstruct、@ PreDestroy等注解就必须声明CommonAnnotationBeanPostProcessor<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>一般来说，这些注解我们还是比较常用，尤其是Antowired的注解，在自动注入的时候更是经常使用，所以如果总是需要按照传统的方式一条一条配置显得有些繁琐和没有必要，于是spring给我们提供<context:annotation-config/>的简化配置方式，自动帮你完成声明。

不过，在如上实现了对bean的定义和注入方式之后，还需要指定那些bean会被注入，这时我们使用注解一般都会配置扫描包路径选项<context:component-scan> 该配置项其实也包含了自动注入上述processor的功能，因此当使用 <context:component-scan/> 后，就可以将 <context:annotation-config/> 移除了

## 9、属性的装配

属性的装配：

普通属性：

@Value(value="itcast")

private String info;

对象属性注入：

* 使用@Autowired 进行自动注入
* @Service 标注业务类
* @Repository 标注DAO
* @Autowired 默认按照类型进行注入
  + 如果存在两个相同Bean类型相同，则按照名称注入（Bean相同指的是实现或继承同一接口）
* @Autowired注入时可以针对成员变量或者setter方法

对象名称注入：

@Autowired

@Qualifier("orderDao")

private OrderDao orderDao;

初始化和销毁方法的注解：

* @PostConstruct 初始化
* @PreDestroy 销毁

Bean的作用范围：

Single （单例，默认）

Prototyle （多例）

Request （相当于request.setAttribute()）

Session （在session范围内）

使用注解@Scope()

## AOP编程

AOP（Aspect Oriented Programing）面向切面编程

Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码

AOP的底层原理 ： 动态代理

JDK动态代理（需要实现接口）

Cglib动态代理（不需要实现接口，原理是继承原有类）

AOP的术语

Joinpoint(连接点):所谓连接点是指那些被拦截到的点。在spring中,这些点指的是方法,因为spring只支持方法类型的连接点.

Pointcut(切入点):所谓切入点是指我们要对哪些Joinpoint进行拦截的定义.

Advice(通知/增强):所谓通知是指拦截到Joinpoint之后所要做的事情就是通知.通知分为前置通知,后置通知,异常通知,最终通知,环绕通知(切面要完成的功能)

Introduction(引介):引介是一种特殊的通知在不修改类代码的前提下, Introduction可以在运行期为类动态地添加一些方法或Field.

Target(目标对象):代理的目标对象

Weaving(织入):是指把增强应用到目标对象来创建新的代理对象的过程.

spring采用动态代理织入，而AspectJ采用编译期织入和类装在期织入

Proxy（代理）:一个类被AOP织入增强后，就产生一个结果代理类

Aspect(切面): 是切入点和通知（引介）的结合

Spring中的通知：

前置通知 org.springframework.aop.MethodBeforeAdvice

\* 在目标方法执行前实施增强

后置通知 org.springframework.aop.AfterReturningAdvice

\* 在目标方法执行后实施增强

环绕通知 org.aopalliance.intercept.MethodInterceptor

\* 在目标方法执行前后实施增强

异常抛出通知 org.springframework.aop.ThrowsAdvice

\* 在方法抛出异常后实施增强

引介通知 org.springframework.aop.IntroductionInterceptor(课程不讲.)

\* 在目标类中添加一些新的方法和属性

## 10、AspectJ编程

基于XML配置

<aop:config>

<aop:aspect>

<aop:around method=”” pointcut=”execution”>(方法)

</aop:aspect>

</aop:config>

<bean id="cacheInterceptor" class="cn.shop.aop.CacheInterceptor"/>

<aop:config>

<aop:aspect ref="cacheInterceptor">

<aop:around method="doAround" pointcut="execution(\* cn.shop.user.\*Service.get\*(..))"/>

<aop:after method="doAfter" pointcut="execution(\* cn.shop.user.\*Service.add\*(..))"/>

<aop:after method="doAfter" pointcut="execution(\* cn.shop.user.\*Service.update\*(..))"/>

<aop:after method="doAfter" pointcut="execution(\* cn.shop.user.\*Service.delete\*(..))"/>

</aop:aspect>

</aop:config>

基于注解：

在配置文件中定义好切面和被代理类（切面：切点和通知的组合）

以及aspectj自动代理<aop:aspectj—autoproxy/>

切面类需要使用@Aspect表明自己是一个切面类，然后在切面类中的方法中定义好切点

@Before(value=”execution(\*cn.userDao.add\*(..))”) <后面两个点代表参数>

在定义的通知中只有环绕通知是使用ProceedingJoinPoint jp来获取切点的信息，其他的都是使用JoinPoint jp 来获取的，通过获取切点的信息可以方便我们使用原方法中的值，在环绕通知中若通过，jp.proceed();

重点：如果不调用 ProceedingJoinPoint的 proceed方法，那么目标方法就被拦截了

在后置通知中若想有返回值，可以在参数中定义一个Object returnVal

