**1、Spring的IOC实现原理**

XML解析+工厂模式+反射

 (1). IoC（Inversion of Control）是指容器控制程序对象之间的关系，而不是传统实现中，由程序代码直接操控。控制权由应用代码中转到了外部**容器**，控制权的转移是所谓反转。 对于Spring而言，就是由Spring来控制对象的生命周期和对象之间的关系；IoC的实现方式是“**依赖注入**（DI）”。从名字上理解，所谓依赖注入，即组件之间的依赖关系由容器在**运行期**决定，即由容器动态地将某种依赖关系注入到组件之中。

(2). 在Spring的工作方式中，所有的类都会在spring容器中登记，告诉spring这是个什么东西，你需要什么东西，然后spring会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

(3). 在系统运行中，动态的向某个对象提供它所需要的其他对象。

(4). 依赖注入的思想是通过反射机制实现的，在实例化一个类时，它通过反射调用类中set方法将事先保存在HashMap中的类属性注入到类中。 总而言之，在传统的对象创建方式中，通常由调用者来创建被调用者的实例，而在Spring中创建被调用者的工作由Spring来完成，然后注入调用者，即所谓的依赖注入or控制反转。 注入方式有两种：依赖注入和设置注入； IoC的优点：降低了组件之间的耦合，降低了业务对象之间替换的复杂性，使之能够灵活的管理对象。

第一步：配置xml文件

 <bean id="dic" class="com.zhy.springIoc.model.Dic"></bean>

第二步：创建工厂类 使用 dom4j解析配置+反射



**2、Spring的AOP实现原理**

动态代理

1. 接口 --定义操作

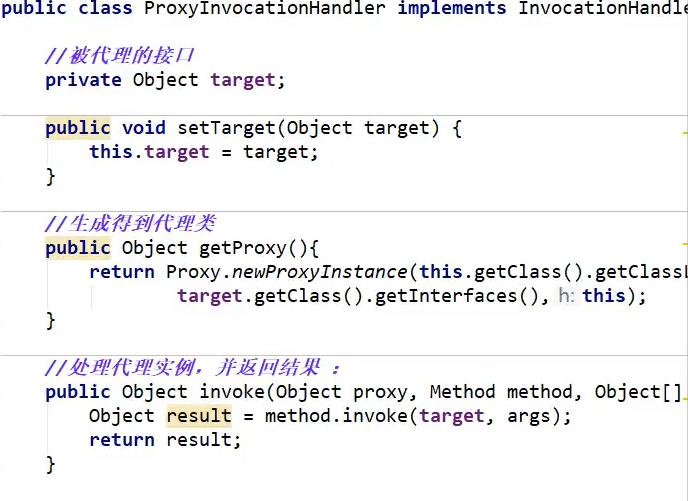
2. 真实角色 --实现操作

3. 代理角色 --调用真实角色实现操作、附加操作

4. 客户 --访问代理角色

动态代理：自动生成代理类，利用反射，getClass得到代理类，invoke执行方法

java动态代理： 利用反射机制生成一个实现代理接口的匿名类，在调用具体方法前调用InvokeHandler来处理。  
cglib动态代理： 利用asm开源包，对代理对象类的class文件加载进来，通过修改其字节码生成子类来处理。



**1、IOC创建对象**

<Bean>： 写在XML中

@Component： 组件，放在类上，说明类被Spring管理了

**2、依赖注入（DI）**

构造器注入

set注入

扩展注入（p命名空间-属性，c命名空间-构造器）

**3、Bean作用域（Scope）**

1）singleton （单一实例）

此取值时表明容器中创建时只存在一个实例，所有引用此bean都是单一实例。

此外，singleton类型的bean定义从容器启动到第一次被请求而实例化开始，只要容器不销毁或退出，该类型的bean的单一实例就会一直存活，典型单例模式，如同servlet在web容器中的生命周期。

2）prototype

spring容器在进行输出prototype的bean对象时，会每次都重新生成一个新的对象给请求方，虽然这种类型的对象的实例化以及属性设置等工作都是由容器负责的，但是只要准备完毕，并且对象实例返回给请求方之后，容器就不在拥有当前对象的引用，请求方需要自己负责当前对象后继生命周期的管理工作，包括该对象的销毁。

3）request

Spring容器会为每个HTTP请求创建一个全新的RequestPrecessor对象，当请求结束后，该对象的生命周期即告结束，**如同java web中request的生命周期**。

4）session

对于web应用来说，放到session中最普遍的就是用户的登录信息，

Spring容器会为每个独立的session创建属于自己的全新的UserPreferences实例，比request scope的bean会存活更长的时间，其他的方面没区别

5）global session

global session只有应用在基于porlet的web应用程序中才有意义，它映射到porlet的global范围的session，如果普通的servlet的web 应用中使用了这个scope，容器会把它作为普通的session的scope对待。

**4、自动装配**

byName：根据id找Bean

byType：根据类型找Bean

**5、使用注解实现自动装配**

@Autowired 通过byType实现，且对象必须存在

@Resource 默认byName，找不到则用byType

@Component 组件，放在类上，说明类被Spring管理了

--dao 【@Repository】

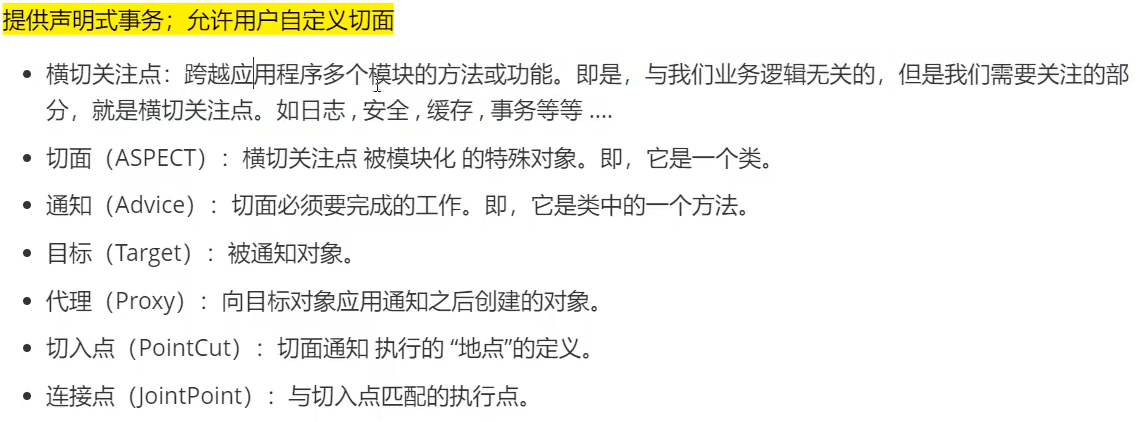
--service 【@Service】

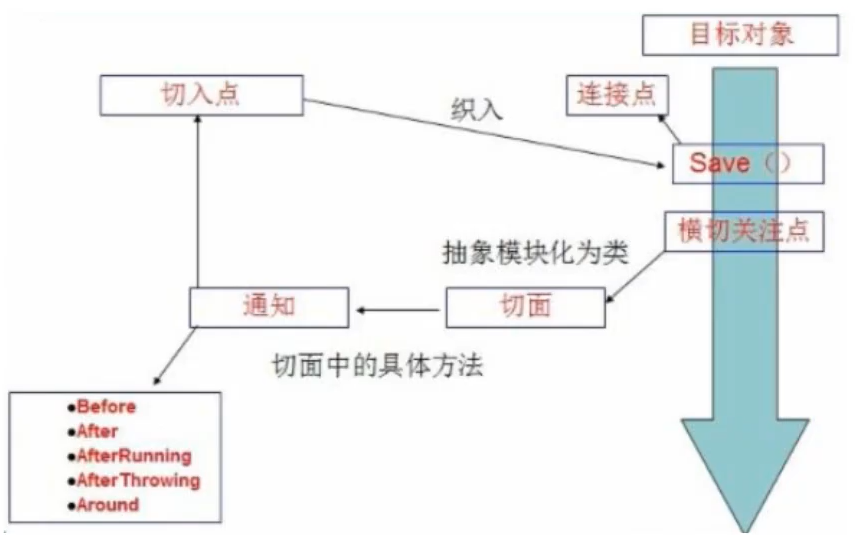
--controller 【@Controller】

**6、Java方式实现自动装配**



**8、Spring使用AOP**

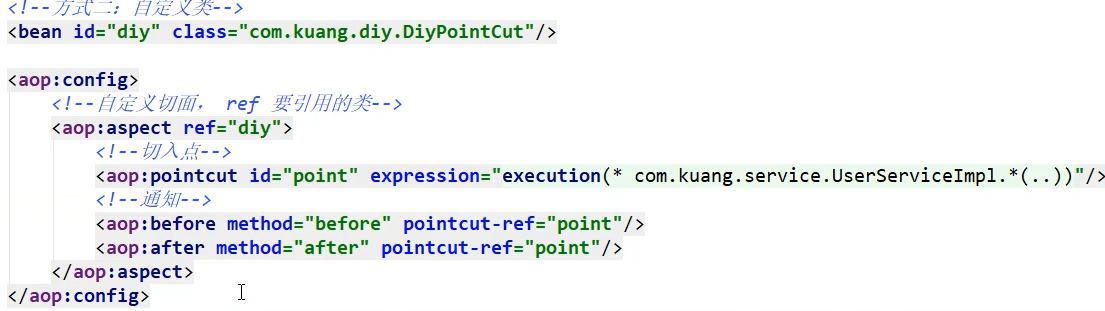




**1. 使用原生Spring API接口**



**2. 自定义类（切面Aspect）实现AOP**



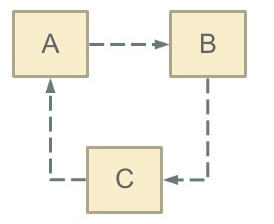
**3. 使用注解实现AOP**



**9、Spring解决循环依赖**

循环依赖：就是N个类循环（嵌套）引用。

通俗的讲就是N个Bean互相引用对方，最终形成闭环。用一副经典的图示可以表示成这样（A、B、C都代表对象，虚线代表引用关系）：



@Service

public class A {

@Autowired

private B b;

}

@Service

public class B {

@Autowired

private A a;

}

**解决方法：三级缓存**

----singletonObjects：用于存放完全初始化好的 bean，从该缓存中取出的 bean 可以直接使用

----earlySingletonObjects：提前曝光的单例对象的cache，存放原始的 bean 对象（尚未填充属性），用于解决循环依赖

----singletonFactories：单例对象工厂的cache，存放 bean 工厂对象，用于解决循环依赖

1、先从一级缓存singletonObjects中去获取。（如果获取到就直接return）

2、如果获取不到或者对象正在创建中（isSingletonCurrentlyInCreation()），那就再从二级缓存earlySingletonObjects中获取。（如果获取到就直接return）

3、如果还是获取不到，且允许singletonFactories（allowEarlyReference=true）通过getObject()获取。就从三级缓存singletonFactory.getObject()获取。

**10、Spring常用注解**

**1.声明bean的注解**

@Component 组件，没有明确的角色

@Service 在业务逻辑层使用（service层）

@Repository 在数据访问层使用（dao层）

@Controller 在展现层使用，控制器的声明（controller层）

**2.注入bean的注解**

@Autowired：由Spring提供

@Inject：由JSR-330提供

@Resource：由JSR-250提供  
@Qualifier 与 @Primary 都可以用于bean的歧义性问题

**3.java配置类相关注解**

@Configuration 声明当前类为配置类，相当于xml形式的Spring配置

@Bean 注解在方法上，声明当前方法的返回值为一个bean，替代xml中的方式

@Configuration 声明当前类为配置类，其中内部组合了@Component注解，表明这个类是一个bean

@ComponentScan 用于对Component进行扫描，相当于xml中的context:component-scan/（类上）

**4.切面（AOP）相关注解**

@Aspect 声明一个切面（类上）  
使用@After、@Before、@Around定义建言（advice），可直接将拦截规则（切点）作为参数。

@After 在方法执行之后执行  
@Before 在方法执行之前执行  
@Around 在方法执行之前与之后执行

@PointCut 声明切点

**5.@Bean的属性支持**

@Scope 设置Spring容器如何新建Bean实例（方法上，得有@Bean）  
其设置类型包括：

Singleton （单例,一个Spring容器中只有一个bean实例，默认模式）,  
Protetype （每次调用新建一个bean）,  
Request （web项目中，给每个http request新建一个bean）,  
Session （web项目中，给每个http session新建一个bean）,  
GlobalSession（给每一个 global http session新建一个Bean实例）

@PostConstruct 由JSR-250提供，在构造函数执行完之后执行，等价于xml配置文件中bean的initMethod

@PreDestory 由JSR-250提供，在Bean销毁之前执行，等价于xml配置文件中bean的destroyMethod

**6.@Value注解**

@Value 为属性注入值（属性上）  
支持如下方式的注入：  
》注入普通字符

@Value("Michael Jackson")

String name;

》注入操作系统属性

@Value("#{systemProperties['os.name']}")

String osName;

》注入表达式结果

@Value("#{ T(java.lang.Math).random() \* 100 }")

String randomNumber;

》注入其它bean属性

@Value("#{domeClass.name}")

String name;

》注入文件资源

@Value("classpath:com/hgs/hello/test.txt")

String Resource file;

》注入网站资源

@Value("http://www.cznovel.com")

Resource url;

》注入配置文件

@Value("${book.name}")

String bookName;

注入配置使用方法：  
① 编写配置文件（test.properties）

② @PropertySource 加载配置文件(类上)

@PropertySource("classpath:com/hgs/hello/test/test.propertie")

**11、Servlet工作流程**

1、当客户端第一次访问该servlet时，服务器创建这个servlet实例对象；

2、紧接着调用servlet的init方法完成这个servlet对象的初始化；

3、服务器创建代表请求的request对象，和代表响应的response对象，调用servlet的service方法，响应客户端请求；

4、service方法执行，向代表响应的response对象中写入将要回送给浏览器的数据。

5、服务器从response对象中取出相应的数据，构建一个http响应，回写给客户机浏览器。

6、浏览器接受响应数据，解析数据进行显示。

**12、Spring事务传播**

事务传播行为（propagation behavior）指的就是当一个事务方法被另一个事务方法调用时，这个事务方法应该如何进行。  
例如：methodA事务方法调用methodB事务方法时，methodB是继续在调用者methodA的事务中运行呢，还是为自己开启一个新事务运行，这就是由methodB的事务传播行为决定的。

**PROPAGATION\_REQUIRED**  
Spring默认的传播机制，能满足绝大部分业务需求，如果外层有事务，则当前事务加入到外层事务，一块提交，一块回滚。如果外层没有事务，新建一个事务执行

**PROPAGATION\_REQUES\_NEW**  
该事务传播机制是每次都会新开启一个事务，同时把外层事务挂起，当当前事务执行完毕，恢复上层事务的执行。如果外层没有事务，执行当前新开启的事务即可

**PROPAGATION\_SUPPORT**  
如果外层有事务，则加入外层事务，如果外层没有事务，则直接使用非事务方式执行。完全依赖外层的事务

**PROPAGATION\_NOT\_SUPPORT**  
该传播机制不支持事务，如果外层存在事务则挂起，执行完当前代码，则恢复外层事务，无论是否异常都不会回滚当前的代码

**PROPAGATION\_NEVER**  
该传播机制不支持外层事务，即如果外层有事务就抛出异常

**PROPAGATION\_MANDATORY**  
与NEVER相反，如果外层没有事务，则抛出异常

**PROPAGATION\_NESTED**  
该传播机制的特点是可以保存状态保存点，当前事务回滚到某一个点，从而避免所有的嵌套事务都回滚，即各自回滚各自的，如果子事务没有把异常吃掉，基本还是会引起全部回滚的。