IOC(控制反转)

将对象的创建权反转给了Spring

作用：接口和实现类之间有较高的耦合，联系过紧，每次都需要修改源代码。

Ioc通过 **工厂模式+反射+配置文件**可以实现解耦，不必每次都要去修改源码就可以实现程序的扩展。

1.底层原理使用技术

- xml配置文件
- dom4j解决xml
- 工厂设计模式
- 反射

2.Spring的bean管理（xml，通过配置文件来创建对象）

1）bean实例化的方式：使用类的无参数构造器创建（重点），但当类里面没有无参数构造器时会出现异常

使用静态工厂创建（创建类的静态方法，返回类对象）

使用实例工厂创建（创建非静态方法，返回类对象）

2）bean标签常用属性：id属性：名称任意，但是不能包含特殊符号，根据id值可以得到配置对象

class属性：创建对象所在类的全路径

name（已不用）：功能和id属性一样，但name属性里面可以包含特殊符号

scope属性：声明bean的作用域

3）bean的作用域（scope的取值）：singleton：默认值，单例

prototye:多例

request:创建对象把对象放到request域里

session：创建对象把对象放到session域里面

glabalSession：创建对象把对象放到globalSession里面

4）依赖注入（属性注入）：set方法注入

有参数构造注入

5）IOC与DI区别：IOC:控制反转，把创建对象交给Spring进行配置

DI:依赖注入，向类里面的属性设置值

依赖注入不能单独存在，需要在ioc基础上完成操作

3.Spring的bean管理（注解方式）

注解：@注解名称（属性名称=属性值），可以使用在类、方法和属性上面

1）创建对象的注解：@Component　WEB层：@Controller 业务层：@Service　持久层：@Repository

2)bean的作用范围(单例还是多例）：@Scope

3）注入属性：@Autowired(在属性上面添加，无需set方法)

@Resource(name="所创建对象所在类中注解的value值")

4.Spring的Bean加载机制：

比如我们这里定义了一个IOC容器，BeanFactroy的子类ClassXMLPathApplicationContext,在他的构造函数中我们会把xml路径写进去以此完成资源定位步骤，接下来就是BeanDefiniton的载入，在构造函数当中有一个refresh()的函数，这个就是载入BeanDefinition的接口，这个方法进去之后是一个同步代码块，把之前的容器销毁和关闭并且创建了一个BeanFatroy，就像对我们的容器重新启动一样，然后我们对BeanDefiniton载入和解析解析完毕之后会把beanDefinition和beanName放入BeanFactory的HashMap中维护。在这里Bean已经被创建完成，然后我们就像IOC容器索要Bean，如果是第一次索要会触发依赖注入，会递归的调用gebBean实现依赖出入。

5.依赖注入的三种方式：

- 接口注入:接口注入的意思是通过接口来实现信息的注入，而其它的类要实现该接口时，就可以实现了注入
- 构造器依赖注入：构造器依赖注入在容器触发构造器的时候完成，该构造器有一系列的参数，每个参数代表注入的对象。

- Setter方法依赖注入：首先容器会触发一个无参构造函数或无参静态工厂方法实例化对象，之后容器调用bean中的setter方法完成Setter方法依赖注入。

6.与new一个对象的区别

spring实现了对象池，一些对象创建和使用完毕之后不会被销毁，放进对象池（某种集合）以备下次使用，下次再需要这个对象，不new，直接从池里出去来用。节省时间，节省cpu。

7.Bean的作用域:

singleton：单例模式，Spring **IoC容器**中**只**会存在一个共享的Bean实例，无论有多少个Bean引用它，始终指向同一对象。Singleton作用域是Spring中的缺省作用域，也可以显示的将Bean定义为singleton模式。

prototype:原型模式，每次**通过**Spring容器获取prototype定义的bean时，容器都将创建一个新的Bean实例，每个Bean实例都有自己的属性和状态，而singleton全局只有一个对象。

根据经验，对有状态的bean使用prototype作用域，而对无状态的bean使用singleton作用域。

request：在**一**次Http请求中，容器会返回该Bean的同一实例。而对不同的Http请求则会产生新的Bean，而且该bean仅在当前Http Request内有效。<bean id="loginAction" class="com.cnblogs.Login" scope="request"/>，针对每一次Http请求，Spring容器根据该bean的定义创建一个全新的实例，且该实例仅在当前Http请求内有效，而其它请求无法看到当前请求中状态的变化，当当前Http请求结束，该bean实例也将会被销毁。

session：在一次Http Session中，容器会返回该Bean的同一实例。而对不同的Session请求则会创建新的实例，该bean实例仅在当前Session内有效。<bean id="userPreference" class="com.ioc.UserPreference" scope="session"/>,同Http请求相同，每一次session请求创建新的实例，而不同的实例之间不共享属性，且实例仅在自己的session请求内有效，请求结束，则实例将被销毁。

global Session：在一个全局的Http Session中，容器会返回该Bean的同一个实例，仅在使用portlet context时有效。

8.Bean的生命周期

实例化：Spring通过new关键字将一个**Bean进行实例化。**

填入属性：Spring将**值和bean引用注入**到bean 的属性中。

- 如果Bean实现了**BeanNameAware**接口，工厂调用Bean的setBeanName()方法传递Bean的ID。
- 如果Bean实现了**BeanFactoryAware**接口，工厂调用setBeanFactory()方法传入工厂自身。
- 如果实现了**ApplicationContextAware**, spring将调用setApplicationContext()方法，将bean所在的上下文的引用进来。
- 如果**BeanPostProcessor**和Bean关联，那么它们的**postProcessBeforeInitialization()**方法将被调用。
- 如果Bean指定了init-method方法，它将被调用。
- 如果有**BeanPostProcessor**和Bean关联，那么它们的postProcessAfterInitialization()方法将被调用
- 最后如果配置了destroy-method方法则注册**DisposableBean**.

使用：到这个时候，Bean已经可以被应用系统使用了，并且将被保留在Bean Factory中直到它不再需要。

销毁：如果Bean实现了DisposableBean接口，就调用其destroy方法。有两种方法可以把它从BeanFactory中删除掉：

- 如果Bean实现了DisposableBean接口，destory()方法被调用。
- 如果指定了订制的销毁方法，就调用这个方法。destory-method()配置时指定。

对几个重要接口的解释：

- **BeanNameAware**: 实现该接口可以获得本身bean的id属性，获得在配置文件中定义好的Bean的ID名

- **BeanFactoryAware**：实现这个接口的bean其实是希望知道自己属于哪一个BeanFactory, 是哪个BeanFactory创建的。
- **ApplicationContextAware**：当一个类实现了这个接口之后，这个类就可以方便地获得 ApplicationContext 中的所有bean。换句话说，就是这个类可以直接获取Spring配置文件中，所有有引用到的bean对象。
- **BeanPostProcessor**是Spring中定义的一个接口，其与InitializingBean和DisposableBean接口类似，也是供Spring进行回调的，Spring将在初始化bean前后对BeanPostProcessor实现类进行回调，Spring容器通过BeanPostProcessor给了我们一个机会对Spring管理的bean进行再加工。比如：我们可以修改bean的属性，可以给bean生成一个动态代理实例等等。

9.BeanFactory 和 ApplicationContext 有什么区别

- Bean工厂(BeanFactory)是Spring框架最核心的接口，提供了高级Ioc的配置机制.
- 应用上下文(ApplicationContext)建立在BeanFacotry基础之上，提供了更多面向应用的功能，如国际化，属性编辑器，事件等等.
- beanFactory是spring框架的基础设施，是面向spring本身，ApplicationContext是面向使用Spring框架的开发者，几乎所有场合都会用到ApplicationContext。