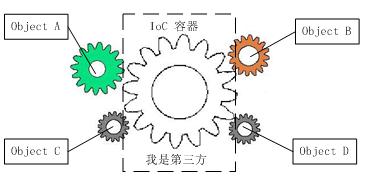
Java IOC容器

IOC(控制反转，Inversion of Control)是面向对象编程的重要法则，用于减少程序模块的耦合问题，也是轻量级Spring框架的核心。IoC的核心目标是通过简单的机制来解决组件依赖的问题，并且在依赖对象的生命周期中对他们进行管理。控制反转一般分为两种类型：依赖注入（DI）和依赖查找：

* 依赖注入，容器全权负责组件装配，把对象传递给需要的对象
* 依赖查找，容器中的对象通过容器的API来查找自己所需的资源和协作对象

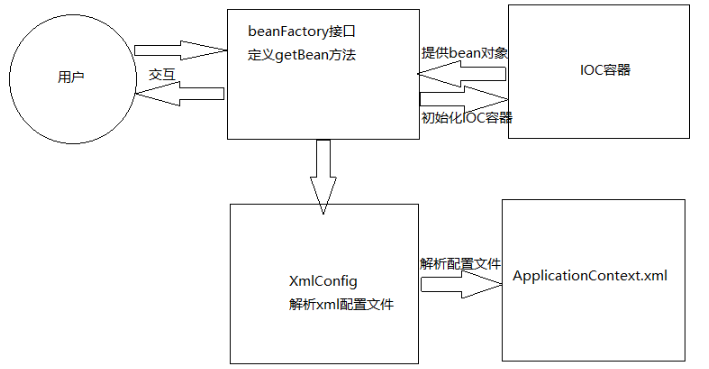
一般使用DI，耦合性地而且组件不会使用到某个容器特定API，可以脱离容器使用。其原理如下图所示：



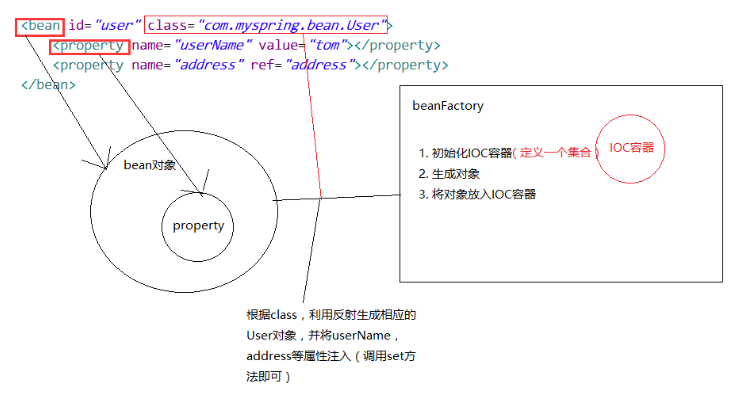
借助第三方（IoC容器）实现具有依赖关系的对象之间的解耦，对象的控制权由IoC容器完成：如果没有IoC容器，对象A依赖对象B，那么在初始化或者运行到某阶段，就需要创建对象B，控制权在对象A；使用IoC容器后对象A与对象B之间没有直接联系，当需要使用对象B时，IoC容器会主动创建对象B注入到对象A需要的地方。对象A获得依赖对象B的过程，由主动行为变成被动行为，这就是控制反转概念的由来，获得依赖对象的过程由自身管理变成了由IoC容器的主动注入。

目前Sun ONE技术体系下的IoC容器包括轻量级容器，Spring、Guice、Pico Container、Avalon等；重量级的有EJB等；其他的JBoss、Jdon等。

下面实现一个简单的IoC容器，架构图解如下：



1）基本思路，如下图：



* 解析xml配置文件
* 根据配置生成相应的对象
* 将对象存入IoC容器

1. 实现

使用dom4j.jar和jaxen.jar来解析xml文件，通过java反射机制生成对象

* 定义Bean类

|  |  |
| --- | --- |
| *@Setter*  *@Getter*  *@ToString*  *public class Address {*  *private String city;*  *}* | *@Getter*  *@Setter*  *@ToString*  *public class User {*  *private String userName;*  *private Address address;*  *}* |

* 创建配置文件application-context.xml，用于初始化Bean

*<?xml version="1.0" encoding="UTF-8"?>*

*<beans>*

*<bean id="address" class="com.fys.myspring.bean.Address">*

*<property name="city" value="fuzhou"></property>*

*</bean>*

*<bean id="user" class="com.fys.myspring.bean.User">*

*<property name="userName" value="tom"></property>*

*<property name="address" ref="address"></property>*

*</bean>*

*</beans>*

* XML配置文件的解析
* 定义配置映射类，Bean和Property

|  |  |
| --- | --- |
| *@Getter*  *@Setter*  *@ToString*  *public class Bean {*  *private String id;*  *private String className;*  *private List<Property> properties = new ArrayList<Property>();*  *}* | *@Getter*  *@Setter*  *@ToString*  *public class Property {*  *private String name;*  *private String value;*  *private String ref;*  *}* |

* XMLConfig，将XML配置映射为上面的参数类

*public class XmlConfig {*

*public static Map<String,Bean> getConfig(String path) {*

*Map<String,Bean> configMap = new HashMap<String, Bean>();*

*Document doc = null;*

*SAXReader reader = new SAXReader();*

*try {*

*InputStream in = new FileInputStream(new File(path));*

*doc = reader.read(in);*

*} catch (Exception e) {*

*e.printStackTrace();*

*throw new RuntimeException("Check XmlConfig Path");*

*}*

*String xpath = "//bean";*

*List<Element> list = doc.selectNodes(xpath);*

*if (list != null) {*

*for(Element beanEle: list) {*

*Bean bean = new Bean();*

*String id = beanEle.attributeValue("id");*

*bean.setId(id);*

*bean.setClassName(beanEle.attributeValue("class"));*

*List<Element> proList = beanEle.elements("property");*

*if(proList != null) {*

*for(Element proEle: proList) {*

*Property prop = new Property();*

*prop.setName(proEle.attributeValue("name"));*

*prop.setValue(proEle.attributeValue("value"));*

*prop.setRef(proEle.attributeValue("ref"));*

*bean.getProperties().add(prop);*

*}*

*}*

*if(configMap.containsKey(id)) {*

*throw new RuntimeException("Bean redefined in Config:" + id);*

*}*

*configMap.put(id, bean);*

*}*

*}*

*return configMap;*

*}*

*}*

* 将配置初始化Bean对象

*public interface BeanFactory {*

*Object getBean(String beanName);*

*}*

实现类为：

*public class ClassPathXmlApplicationContext implements BeanFactory {*

*private Map<String, Object> ioc;*

*private Map<String, Bean> config;*

*public ClassPathXmlApplicationContext(String path) {*

*ioc = new HashMap<String, Object>();*

*config = XmlConfig.getConfig(path);*

*if(config != null) {*

*for(Map.Entry<String,Bean> entry: config.entrySet()) {*

*String beanId = entry.getKey();*

*Bean bean = entry.getValue();*

*Object object = createBean(bean);*

*System.out.println("IOC put:" + beanId + " => "+ object.toString());*

*ioc.put(beanId, object);*

*}*

*}*

*}*

*private Object createBean(Bean bean) {*

*try {*

*String beanId = bean.getId();*

*String className = bean.getClassName();*

*Class c = null;*

*Object object = null;*

*c = Class.forName(className);*

*object = c.newInstance();*

*if (bean.getProperties() != null) {*

*for (Property p : bean.getProperties()) {*

*if (p.getValue() != null) {*

*Method getMethod = BeanUtil.getSetterMethod(object, p.getName());*

*getMethod.invoke(object, p.getValue());*

*}*

*if(p.getRef() != null) {*

*Method getMethod = BeanUtil.getSetterMethod(object, p.getName());*

*Object obj = ioc.get(p.getRef());*

*if(obj == null) {*

*throw new RuntimeException("Cann't find dependency:" + p.getRef());*

*} else {*

*getMethod.invoke(object, obj);*

*}*

*}*

*}*

*}*

*return object;*

*} catch (Exception e) {*

*throw new RuntimeException("Config Not Illegal" ,e);*

*}*

*}*

*public Object getBean(String beanName) {*

*return ioc.get(beanName);*

*}*

*}*

* BeanUtil，根据属性值获取Setter方法进行Bean的初始化

*public class BeanUtil {*

*public static Method getSetterMethod(Object obj, String name) {*

*Method method = null;*

*name = "set"+name.substring(0,1).toUpperCase()+name.substring(1);*

*try {*

*Method[] methods = obj.getClass().getMethods();*

*//遍历该类的所有方法*

*for(int i=0;i<methods.length;i++){*

*Method m = methods[i];*

*if(m.getName().equals(name)){*

*method = obj.getClass().getMethod(name,m.getParameterTypes());*

*break;*

*}*

*}*

*} catch (NoSuchMethodException e) {*

*e.printStackTrace();*

*}*

*return method;*

*}*

*}*

1. 测试类

*private static void testIOC() {*

*BeanFactory bf = new ClassPathXmlApplicationContext("/root/workspace/myspring/src/main/resources/application-context.xml");*

*User user = (User) bf.getBean("user");*

*System.out.println("User Bean:" + user.toString());*

*Address address = (Address) bf.getBean("address");*

*System.out.println("Address Bean:" + address.toString());*

*System.out.println("User's Address HashCode: "+ user.getAddress().hashCode());*

*System.out.println("Address Bean: " + address.hashCode());*

*}*

程序输出入下：

*User Bean:User(userName=tom, address=Address(city=fuzhou))*

*Address Bean:Address(city=fuzhou)*

*User's Address HashCode: 1993134103*

*Address Bean: 1993134103*

https://blog.csdn.net/u010837612/article/details/50686573