# [spring ioc原理（看完后大家可以自己写一个spring）](http://blog.csdn.net/it_man/article/details/4402245)

[控制反转/依赖注入](http://blog.csdn.net/it_man/article/details/blog/262925)

最近，买了本Spring入门书：spring In Action 。大致浏览了下感觉还不错。就是入门了点。Manning的书还是不错的，我虽然不像哪些只看Manning书的人那样专注于Manning,但怀着崇敬的心情和激情通览了一遍。又一次接受了IOC 、DI、AOP等Spring核心概念。 先就IOC和DI谈一点我的看法。  
  
IOC（DI）：其实这个Spring架构核心的概念没有这么复杂，更不像有些书上描述的那样晦涩。java程序员都知道：java程序中的每个业务逻辑至少需要两个或以上的对象来协作完成，通常，每个对象在使用他的合作对象时，自己均要使用像new object（） 这样的语法来完成合作对象的申请工作。你会发现：对象间的耦合度高了。而IOC的思想是：Spring容器来实现这些相互依赖对象的创建、协调工作。对象只需要关系业务逻辑本身就可以了。从这方面来说，对象如何得到他的协作对象的责任被反转了（IOC、DI）。  
  
这是我对Spring的IOC的体会。DI其实就是IOC的另外一种说法。DI是由Martin Fowler 在2004年初的一篇论文中首次提出的。他总结：控制的什么被反转了？就是：获得依赖对象的方式反转了。  
  
如果对这一核心概念还不理解：这里引用一个叫Bromon的blog上找到的浅显易懂的答案：

IoC与DI

　　首先想说说IoC（Inversion of Control，控制倒转）。这是spring的核心，贯穿始终。所谓IoC，对于spring框架来说，就是由spring来负责控制对象的生命周期和对象间的关系。这是什么意思呢，举个简单的例子，我们是如何找女朋友的？常见的情况是，我们到处去看哪里有长得漂亮身材又好的mm，然后打听她们的兴趣爱好、qq号、电话号、ip号、iq号………，想办法认识她们，投其所好送其所要，然后嘿嘿……这个过程是复杂深奥的，我们必须自己设计和面对每个环节。传统的程序开发也是如此，在一个对象中，如果要使用另外的对象，就必须得到它（自己new一个，或者从JNDI中查询一个），使用完之后还要将对象销毁（比如Connection等），对象始终会和其他的接口或类藕合起来。

　　那么IoC是如何做的呢？有点像通过婚介找女朋友，在我和女朋友之间引入了一个第三者：婚姻介绍所。婚介管理了很多男男女女的资料，我可以向婚介提出一个列表，告诉它我想找个什么样的女朋友，比如长得像李嘉欣，身材像林熙雷，唱歌像周杰伦，速度像卡洛斯，技术像齐达内之类的，然后婚介就会按照我们的要求，提供一个mm，我们只需要去和她谈恋爱、结婚就行了。简单明了，如果婚介给我们的人选不符合要求，我们就会抛出异常。整个过程不再由我自己控制，而是有婚介这样一个类似容器的机构来控制。Spring所倡导的开发方式就是如此，所有的类都会在spring容器中登记，告诉spring你是个什么东西，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。如果你还不明白的话，我决定放弃。

IoC的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的。比如对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。A需要依赖 Connection才能正常运行，而这个Connection是由spring注入到A中的，依赖注入的名字就这么来的。那么DI是如何实现的呢？ Java 1.3之后一个重要特征是反射（reflection），它允许程序在运行的时候动态的生成对象、执行对象的方法、改变对象的属性，spring就是通过反射来实现注入的。关于反射的相关资料请查阅java doc。  
　理解了IoC和DI的概念后，一切都将变得简单明了，剩下的工作只是在spring的框架中堆积木而已。  
  
如果还不明白，放弃java吧！

**下面来让大家了解一下Spring到底是怎么运行的。**

Java代码

1. **public static void main(String[] args) {**
2. **ApplicationContext context = new FileSystemXmlApplicationContext(**
3. **"applicationContext.xml");**
4. **Animal animal = (Animal) context.getBean("animal");**
5. **animal.say();**
6. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **public static void main(String[] args) {**
2. **ApplicationContext context = new FileSystemXmlApplicationContext(**
3. **"applicationContext.xml");**
4. **Animal animal = (Animal) context.getBean("animal");**
5. **animal.say();**
6. **}**

**这段代码你一定很熟悉吧，不过还是让我们分析一下它吧，首先是applicationContext.xml**

Java代码

1. **<bean id="animal" class="phz.springframework.test.Cat">**
2. **<property name="name" value="kitty" />**
3. **</bean>**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **<bean id="animal" class="phz.springframework.test.Cat">**
2. **<property name="name" value="kitty" />**
3. **</bean>**

**他有一个类phz.springframework.test.Cat**

Java代码

1. **public class Cat implements Animal {**
2. **private String name;**
3. **public void say() {**
4. **System.out.println("I am " + name + "!");**
5. **}**
6. **public void setName(String name) {**
7. **this.name = name;**
8. **}**
9. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **public class Cat implements Animal {**
2. **private String name;**
3. **public void say() {**
4. **System.out.println("I am " + name + "!");**
5. **}**
6. **public void setName(String name) {**
7. **this.name = name;**
8. **}**
9. **}**

**实现了phz.springframework.test.Animal接口**

Java代码

1. **public interface Animal {**
2. **public void say();**
3. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **public interface Animal {**
2. **public void say();**
3. **}**

**很明显上面的代码输出I am kitty!   
  
那么到底Spring是如何做到的呢？   
接下来就让我们自己写个Spring 来看看Spring 到底是怎么运行的吧！   
  
首先，我们定义一个Bean类，这个类用来存放一个Bean拥有的属性**

Java代码

1. **/\* Bean Id \*/**
2. **private String id;**
3. **/\* Bean Class \*/**
4. **private String type;**
5. **/\* Bean Property \*/**
6. **private Map<String, Object> properties = new HashMap<String, Object>();**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **/\* Bean Id \*/**
2. **private String id;**
3. **/\* Bean Class \*/**
4. **private String type;**
5. **/\* Bean Property \*/**
6. **private Map<String, Object> properties = new HashMap<String, Object>();**

**一个Bean包括id,type,和Properties。   
  
接下来Spring 就开始加载我们的配置文件了，将我们配置的信息保存在一个HashMap中，HashMap的key就是Bean 的 Id ，HasMap 的value是这个Bean，只有这样我们才能通过context.getBean("animal")这个方法获得Animal这个类。我们都知道Spirng可以注入基本类型，而且可以注入像List，Map这样的类型，接下来就让我们以Map为例看看Spring是怎么保存的吧   
  
Map配置可以像下面的**

Java代码

1. **<bean id="test" class="Test">**
2. **<property name="testMap">**
3. **<map>**
4. **<entry key="a">**
5. **<value>1</value>**
6. **</entry>**
7. **<entry key="b">**
8. **<value>2</value>**
9. **</entry>**
10. **</map>**
11. **</property>**
12. **</bean>**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **<bean id="test" class="Test">**
2. **<property name="testMap">**
3. **<map>**
4. **<entry key="a">**
5. **<value>1</value>**
6. **</entry>**
7. **<entry key="b">**
8. **<value>2</value>**
9. **</entry>**
10. **</map>**
11. **</property>**
12. **</bean>**

**Spring是怎样保存上面的配置呢？，代码如下：**

Java代码

1. **if (beanProperty.element("map") != null) {**
2. **Map<String, Object> propertiesMap = new HashMap<String, Object>();**
3. **Element propertiesListMap = (Element) beanProperty**
4. **.elements().get(0);**
5. **Iterator<?> propertiesIterator = propertiesListMap**
6. **.elements().iterator();**
7. **while (propertiesIterator.hasNext()) {**
8. **Element vet = (Element) propertiesIterator.next();**
9. **if (vet.getName().equals("entry")) {**
10. **String key = vet.attributeValue("key");**
11. **Iterator<?> valuesIterator = vet.elements()**
12. **.iterator();**
13. **while (valuesIterator.hasNext()) {**
14. **Element value = (Element) valuesIterator.next();**
15. **if (value.getName().equals("value")) {**
16. **propertiesMap.put(key, value.getText());**
17. **}**
18. **if (value.getName().equals("ref")) {**
19. **propertiesMap.put(key, new String[] { value**
20. **.attributeValue("bean") });**
21. **}**
22. **}**
23. **}**
24. **}**
25. **bean.getProperties().put(name, propertiesMap);**
26. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **if (beanProperty.element("map") != null) {**
2. **Map<String, Object> propertiesMap = new HashMap<String, Object>();**
3. **Element propertiesListMap = (Element) beanProperty**
4. **.elements().get(0);**
5. **Iterator<?> propertiesIterator = propertiesListMap**
6. **.elements().iterator();**
7. **while (propertiesIterator.hasNext()) {**
8. **Element vet = (Element) propertiesIterator.next();**
9. **if (vet.getName().equals("entry")) {**
10. **String key = vet.attributeValue("key");**
11. **Iterator<?> valuesIterator = vet.elements()**
12. **.iterator();**
13. **while (valuesIterator.hasNext()) {**
14. **Element value = (Element) valuesIterator.next();**
15. **if (value.getName().equals("value")) {**
16. **propertiesMap.put(key, value.getText());**
17. **}**
18. **if (value.getName().equals("ref")) {**
19. **propertiesMap.put(key, new String[] { value**
20. **.attributeValue("bean") });**
21. **}**
22. **}**
23. **}**
24. **}**
25. **bean.getProperties().put(name, propertiesMap);**
26. **}**

**接下来就进入最核心部分了，让我们看看Spring 到底是怎么依赖注入的吧，其实依赖注入的思想也很简单，它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。让我们看看具体它是怎么做的吧。   
首先实例化一个类，像这样**

Java代码

1. **public static Object newInstance(String className) {**
2. **Class<?> cls = null;**
3. **Object obj = null;**
4. **try {**
5. **cls = Class.forName(className);**
6. **obj = cls.newInstance();**
7. **} catch (ClassNotFoundException e) {**
8. **throw new RuntimeException(e);**
9. **} catch (InstantiationException e) {**
10. **throw new RuntimeException(e);**
11. **} catch (IllegalAccessException e) {**
12. **throw new RuntimeException(e);**
13. **}**
14. **return obj;**
15. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **public static Object newInstance(String className) {**
2. **Class<?> cls = null;**
3. **Object obj = null;**
4. **try {**
5. **cls = Class.forName(className);**
6. **obj = cls.newInstance();**
7. **} catch (ClassNotFoundException e) {**
8. **throw new RuntimeException(e);**
9. **} catch (InstantiationException e) {**
10. **throw new RuntimeException(e);**
11. **} catch (IllegalAccessException e) {**
12. **throw new RuntimeException(e);**
13. **}**
14. **return obj;**
15. **}**

**接着它将这个类的依赖注入进去，像这样**

Java代码

1. **public static void setProperty(Object obj, String name, String value) {**
2. **Class<? extends Object> clazz = obj.getClass();**
3. **try {**
4. **String methodName = returnSetMthodName(name);**
5. **Method[] ms = clazz.getMethods();**
6. **for (Method m : ms) {**
7. **if (m.getName().equals(methodName)) {**
8. **if (m.getParameterTypes().length == 1) {**
9. **Class<?> clazzParameterType = m.getParameterTypes()[0];**
10. **setFieldValue(clazzParameterType.getName(), value, m,**
11. **obj);**
12. **break;**
13. **}**
14. **}**
15. **}**
16. **} catch (SecurityException e) {**
17. **throw new RuntimeException(e);**
18. **} catch (IllegalArgumentException e) {**
19. **throw new RuntimeException(e);**
20. **} catch (IllegalAccessException e) {**
21. **throw new RuntimeException(e);**
22. **} catch (InvocationTargetException e) {**
23. **throw new RuntimeException(e);**
24. **}**
25. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **public static void setProperty(Object obj, String name, String value) {**
2. **Class<? extends Object> clazz = obj.getClass();**
3. **try {**
4. **String methodName = returnSetMthodName(name);**
5. **Method[] ms = clazz.getMethods();**
6. **for (Method m : ms) {**
7. **if (m.getName().equals(methodName)) {**
8. **if (m.getParameterTypes().length == 1) {**
9. **Class<?> clazzParameterType = m.getParameterTypes()[0];**
10. **setFieldValue(clazzParameterType.getName(), value, m,**
11. **obj);**
12. **break;**
13. **}**
14. **}**
15. **}**
16. **} catch (SecurityException e) {**
17. **throw new RuntimeException(e);**
18. **} catch (IllegalArgumentException e) {**
19. **throw new RuntimeException(e);**
20. **} catch (IllegalAccessException e) {**
21. **throw new RuntimeException(e);**
22. **} catch (InvocationTargetException e) {**
23. **throw new RuntimeException(e);**
24. **}**
25. **}**

**最后它将这个类的实例返回给我们，我们就可以用了。我们还是以Map为例看看它是怎么做的，我写的代码里面是创建一个HashMap并把该HashMap注入到需要注入的类中，像这样，**

Java代码

1. **if (value instanceof Map) {**
2. **Iterator<?> entryIterator = ((Map<?, ?>) value).entrySet()**
3. **.iterator();**
4. **Map<String, Object> map = new HashMap<String, Object>();**
5. **while (entryIterator.hasNext()) {**
6. **Entry<?, ?> entryMap = (Entry<?, ?>) entryIterator.next();**
7. **if (entryMap.getValue() instanceof String[]) {**
8. **map.put((String) entryMap.getKey(),**
9. **getBean(((String[]) entryMap.getValue())[0]));**
10. **}**
11. **}**
12. **BeanProcesser.setProperty(obj, property, map);**
13. **}**

**[java]** [view plain](http://blog.csdn.net/it_man/article/details/4402245) [copy](http://blog.csdn.net/it_man/article/details/4402245)

1. **if (value instanceof Map) {**
2. **Iterator<?> entryIterator = ((Map<?, ?>) value).entrySet()**
3. **.iterator();**
4. **Map<String, Object> map = new HashMap<String, Object>();**
5. **while (entryIterator.hasNext()) {**
6. **Entry<?, ?> entryMap = (Entry<?, ?>) entryIterator.next();**
7. **if (entryMap.getValue() instanceof String[]) {**
8. **map.put((String) entryMap.getKey(),**
9. **getBean(((String[]) entryMap.getValue())[0]));**
10. **}**
11. **}**
12. **BeanProcesser.setProperty(obj, property, map);**
13. **}**

**好了，这样我们就可以用Spring 给我们创建的类了，是不是也不是很难啊？当然Spring能做到的远不止这些，这个示例程序仅仅提供了Spring最核心的依赖注入功能中的一部分。**