

# HelloSpring

## 1 HelloSpring

- 编写一个类

```
1 package com.kuang.pojo;
2
3 import java.io.Serializable;
4
5 public class Hello implements Serializable {
6     private String str;
7
8     public String getStr() {
9         return str;
10    }
11
12    public void setStr(String str) {
13        this.str = str;
14    }
15
16    @Override
17    public String toString() {
18        return "Hello{" +
19            "str='" + str + '\'' +
20            '}';
21    }
22 }
```

- 编写配置文件 (这里先起名为beans.xml)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <!--使用Spring来创建对象
8         在Spring中，这些都称为Bean
9         bean = 对象
10        class=Hello ==> new Hello();
11
12        类型 变量名 = new 类型();
13        Hello hello = new Hello();
14
15        id ==> 变量名
16        class ==> new的对象
17        property ==> 相当于给对象中的属性设置一个值（利用set注入，pojo中去掉set会
18        报错）
19        -->
20    <bean id="hello" class="com.kuang.pojo.Hello">
21        <property name="str" value="HelloSpring"/>
22    </bean>
```

- 运行测试

```

1 package com.kuang.test;
2
3 import com.kuang.pojo.Hello;
4 import org.junit.Test;
5 import org.springframework.context.ApplicationContext;
6 import org.springframework.context.support.ClassPathXmlApplicationContext;
7
8 public class TestHello {
9     @Test
10    public void testHello() {
11        //获取Spring的上下文对象
12        ApplicationContext context = new
13        ClassPathXmlApplicationContext("beans.xml");
14
15        //我们的对象现在都在Spring中管理，要使用直接在里面取出来即可
16        Hello hello = (Hello) context.getBean("hello");
17
18        System.out.println(hello.toString());
19
20    }
21
22 }

```

```

Hello{str='HelloSpring'}
Process finished with exit code 0

```

因为，前面的property给Hello类的属性str赋值为“HelloSpring”，所以这里打印出来的str的属性值。

## 2 之前的那个例子

- bean配置

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5       http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <!--都没有属性，不需要property进行注入-->
8     <bean id="mysqlImpl" class="com.kuang.daoImpl.UserDaoMySQLImpl"/>
9     <bean id="oracleImpl" class="com.kuang.daoImpl.UserDaoOracleImpl"/>
10
11     <bean id="userServiceImpl"
12     class="com.kuang.serviceImpl.UserServiceImpl">
13         <!--
14         ref --> 引用Spring容器中创建好的对象
15         value --> 具体的值，基本数据类型
16         -->

```

```
16     <property name="userDao" ref="mysqlImpl"/>
17 </bean>
18
19 </beans>
```

- 运行测试

```
1 @Test
2 public void testSpring() {
3     //拿到Spring的容器
4     ApplicationContext context = new
    ClassPathXmlApplicationContext("beans.xml");
5
6     UserServiceImpl userServiceImpl = (UserServiceImpl)
    context.getBean("userServiceImpl");
7     userServiceImpl.getUser();
8
9 }
```

获取到用户的Mysql数据信息

Process finished with exit code 0

因为配置的ref指向mysqlImpl对应的bean。

- 因此，我们可以根据需求修改配置文件即可，而不用动到源代码。修改配置文件就相对比较容易。

### 3 总结

以上的过程就叫控制反转：

- 控制：谁来控制对象的创建，传统应用程序的对象是由程序本身控制创建的，使用Spring后，对象是由Spring来创建的。
- 反转：程序本身不创建对象，而变成被动的接收对象。
- 依赖注入：就是利用set方法来进行注入的。
- IOC是一种编程思想，由主动的编程变成被动的接收。
- 可以通过newClassPathXmlApplicationContext去浏览一下底层源码。

**OK，到了现在，我们彻底不用再程序中去改动了，要实现不同的操作，只需要在xml配置文件中进行修改，所谓的IOC,一句话搞定：对象由Spring 来创建，管理，装配！**