10 Spring_配置详解_方法注入(了解)

`Spring`

一：概述

> 1、bean A依赖于beanB，而且A需要每次使用B时都是新的B，而不是同一个B
> 2、Spring默认的注入机制仅在创建A时注入一个beanB(创建出所有的bean)，而不是每次使用时注入。这种情况如下处理呢？

二:典型的做法(了解)

> 1、把B配置为scope="prototype"，然后在A中编写一个getB()方法每次从容器中getBean

1、实体类

```
public class Car {
    private String name;
    private String clor;
    public Car() {
        super();
    System.out.println("---Car 执行了-----");
    }
    //对应个的set get toString方法
    }
```

```
public class Users {
    ApplicationContext ac;
    // 注入容器对象
    public void setAC(ApplicationContext ac) {
        this.ac = ac;
    }
```

```
    // 业务方法
    public void doSerivce() {
        Car c = getNewB();
        System.out.println("获取到了Car" + c);
    }
    // 获取Car
    public Car getNewB() {
        return ac.getBean(Car.class);
    }
}
```

2、配置文件

```
<bean id="users" class="com.spring.bean.Users" autowire="byType">
</bean>
<!--users使用的时候才注入Car  -->
<bean id="car" class="com.spring.bean.Car" scope="prototype"></bean>
```

3、测试

```
import javax.annotation.Resource;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner
;
import com.spring.bean.Users;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Test {
    @Resource(name="users")
    private Users  users;
    @org.junit.Test
    public  void getVoid02() throws Exception{
        users.doSerivce();
    }
}
```

```
log4j:WARN No appenders could be found for logger (org.spring
log4j:WARN Please initialize the log4j system properly.
---Car 执行了-----
获取到了CarCar [name=null, clor=null]
```

三:Lookup方法注入

> 1、Spring可以重写bean中的某个方法(这个方法返回值是一个bean)
> 2、然后在容器中需找(lookup)相应的bean，作为返回值。

1、实体类

```
public class Student {
}
```

```
public abstract class Teacher {
/**
 *抽象方法和非抽象方法都可
 */
abstract public Student getStu();
public Student getStu2(){
        return null;
    }
}
```

2、配置

```
<bean id="teacher" class="com.spring.bean.Teacher">
    <lookup-method name="getStu" bean="student" />
    <lookup-method name="getStu2" bean="student" />
</bean>
<bean id="student" class="com.spring.bean.Student"></bean>
```

3、测试

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Test {
    @Resource(name="teacher")
    private Teacher  teacher;
    @org.junit.Test
    public  void getVoid02() throws Exception{
        Student stu = teacher.getStu();
        System.out.println(stu);
    }
}
```

四:方法覆盖实现注入(了解)

> 1、还可以覆盖bean中的某个方法的具体实现

1、实体类

```
public class A {
    public void getA(){
        System.out.println("你好");
    }
}
```

```
import org.springframework.beans.factory.support.MethodReplacer;
public class B  implements MethodReplacer{
    @Override
    public Object reimplement(Object arg0, Method arg1, Object[] arg2)
throws Throwable {
            System.out.println(arg1.getName()+":被执行了，替换其实现");
        return null;
    }
}
```

2、配置

```
<bean id="b" class="com.spring.bean.B">
</bean>
<bean id="a" class="com.spring.bean.A">
<!--name:指定要替换的方法签名
    replacer: 使用bean_a去替换执行-->
    <replaced-method name="getA" replacer="b">
    </replaced-method>
</bean>
```

3、测试

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Test {
    @Resource(name="a")
    private A  a;
    @org.junit.Test
    public  void getVoid02() throws Exception{
       a.getA();
    }
}
```