基于注解的DI：通过注解的方式完成对象创建，属性赋值

使用注解的步骤：

1）、在项目pom文件中加入相应的maven依赖spring-context，当我们把这个依赖的添加会间接加入spring-aop的依赖，使用注解就必须有spring-aop依赖

2）、在需要的类上方加上注解类（不同的注解功能不同）

3）、在spring配置文件中，声明一个组件扫描器，在组件扫描器标签中声明注解在项目中的具体位置，让Spring去获取注解，然后根据注解的功能进行功能实现

需要学习的注解：

```
1、@Component
2、@Respotory
3、@Service
4、@Controller
5、@Value
6、@Autowire
7、@Resource
其中@Component注解的功能是创建对象，2、3、4注解是1注解的分注解，也是可以创建对象的功能，但他们自身都有@Component注解没有的额外功能
```

在idea-maven-spring工程中新建一个项目模块c-spring-di-anno-03，进行基于注解的DI实现测试

1、测试@Component注解和@Respotory、@Service、@Controller

1）、创建一个bao01包，在其中新建一个Student类，在定义类的上方使用注解，具体内容如下所示

```
import org.springframework.stereotype.Component;

/**
 * 注解@Component：
 *         创建对象，相当于配置文件的<bean></bean>功能
 *         其中有个属性value：填的是对象的名称，相当于bean标签中的id
 *         value的值在该项目中是唯一的，创建的对象在Spring容器中只有一个，单例
 *         位置：在类的上面，如下所示
 *
 * Spring中和@Component注解功能一致可创建对象的注解还有：
 * @Reposoitory（用在持久层的类上面）：
```

```
 *         用在dao类上，表示创建dao实现类对象，dao类对象是能够访问数据库的
 * @Service（用在业务逻辑层的类上面）：
 *         放在service类上，创建service实现类对象，这类对象做业务处理，有事务的功能等
 * @Controller（用在控制器上）：
 *         放在处理器（控制器）类上，创建控制器对象，该对象能够接收用户输入参数，显示处理请求结
果
 *  上面三个和@Component注解都能创建对象，但各自都具有额外的功能
 *  使得项目的各个对象分层，声明了该对象是属于什么层对象，有什么特定能力
 * @Component注解一般用在不知道放到什么层中的Java类上面
 * 所以这四个注解并不是功能一模一样的注解
 */
//标准使用该注解写法
//@Component(value = "myStudent")

//学注解的时候知道，注解属性名为value时，且只需要写value时可以省略value
//@Component("myStudent")

//也可以不指定对象的名称，默认命名为首字母小写的类名
@Component
public class Student {

    public Student() {
        System.out.println("Student无参构造方法执行！");
    }

    private String name;
    private int age;

    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

2）、resources目录新建spring配置文件，在其中配置组件扫描器，内容如下

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd">
```

```
    <!--
        当我们添加组件扫描器后上面的配置信息变化：
        1、在xsi:schemaLocation=""的引号中加入了另一个新的约束文件：
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        2、给这个新的约束文件在命名空间起个名称：
        xmlns:context="http://www.springframework.org/schema/context"
        其中名称前面的context表示该约束文件的前缀，由于Spring框架很大，为了避免
        配置文件中标签名重复，在每个约束文件的命名空间前添加一个前缀，来区分是哪
        个约束文件的标签。
        如下面的组件扫描标签前面添加context：就可以知道该标签是来自哪个命名空间了
    -->
    <!--
        声明组件扫描器（component-scan):component的意思是组件，组件就是对象的意思
        该扫描器标签中有一个属性：base-package="xxx"
        表示指定扫描器到那个包中扫描注解的，里面就是包的路径
        配置好包路径后启动项目扫描器就会把这个包和其子包中的所有类进行扫描，看看那个类中有注解
        将有注解的类按照注解的功能创建对象，或者是赋值等等
    -->
    <context:component-scan base-package="com.studymyself.bao01"/>

    <!--三种方式指定多个包
        第一种：使用多个component-scan标签来指定多个包
        第二种：在base-package属性值中填写每个包的路径，每条包路径使用;或，来分隔
        第四种：在base-package属性值指定一个父类包路径，但不要指定根父类，不然会
                由于扫描文件过多，降低项目启动效率
    -->

</beans>
```

新建一个测试类MyTest01.java，测试是否创建了对象

```
package com.studymyself;

import com.studymyself.bao01.Student;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest01 {

    @Test
    public void testStudent(){

        //配置文件路径
        String config = "applicationContext.xml";

        //获取容器对象
        ApplicationContext ac = new ClassPathXmlApplicationContext(config);

        //获取Student类对象
        Student student1 = (Student) ac.getBean("myStudent");
        System.out.println("第一次获取的student1"+student1);
        Student student2 = (Student) ac.getBean("myStudent");
        System.out.println("再次获取的student1"+student2);
```

```
    }

}
```