QFEDU 千锋教育 WWW.QFEDU.COM

第三阶段 Spring-02-注入和注解方式操作

回顾：

```
1. Spring的简介
2. IOC和DI的简介和关系
3. Spring对象创建过程和配置
```

今天任务

QFEDU 千锋教育 WWW.QFEDU.COM

1. 依赖注入详解
2. 使用注解的方式实现依赖注入
3. Spring中使用JUnit测试

教学目标

1. 掌握依赖注入的各种配置和属性
2. 掌握注解的方式实现依赖注入
3. 掌握在Spring中使用JUnit测试

一. 依赖注入

测试类:Person.java

创建配置文件：`applicationContext-injection.xml`

创建测试代码：`InjectionTest.java`

1. set方法注入

1.1 基本类型值注入使用value

配置:

```
  <!-- value值为基本类型 -->
  <bean name="person"  class="com.itqf.spring.bean.Person" >
       <property name="name" value="jeck" />
       <property name="age" value="11"/>
  </bean>
```

测试代码:

```
  @Test
  public  void  test1(){
        //TODO 测试基本数据类型注入数据
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext-injection.xml");

        Person person = context.getBean("person", Person.class);

        System.out.println("person = " + person);
        //输出结果:------------> Person.Person
        //        person = Person{name='jeck', age=11}
 }
```

QFEDU 千锋教育 WWW.QFEDU.COM

1.2 引入类型值注入ref

创建 Car.java:

```
public class Car {
    private String name;
    private String color;

    public Car() {
        super();
        System.out.println("Car的空参构造方法");
    }
    //getter、setter、toString
}
```

修改Person.java,在Person中引入Car:

```
public class Person {

    private String name;
    private Integer age;
    private Car car;
    //构造方法 getter setter toString方法
}
```

配置:利用ref属性给 person的car属性赋值

```
  <bean name="person1" class="com.itqf.spring.bean.Person">
      <property name="name" value="helen"></property>
      <property name="age" value="18"></property>
      <property name="car" ref="car"></property>
  </bean>

  <bean name="car" class="com.itqf.spring.bean.Car">
      <property name="name" value="MINI"></property>
      <property name="color" value="灰色" ></property>
  </bean>
```

测试: 使用之前测试用例即可!

2.构造函数注入

2.1 单个有参构造方法注入

QFEDU 千锋教育 WWW.QFEDU.COM

在Person中创建有参构造函数:

```
 public  Person(String  name , Car car){
        this.name = name;
        this.car = car;
        System.out.println("Person的有参构造方法:"+name+car);
 }
```

配置:

```
<bean name="person" class="com.itqf.spring.bean.Person">
           <constructor-arg name="name" value="rose"/>
           <constructor-arg name="car"  ref="car"/>
 </bean>
 <!-- 构造函数car时候引入 -->
 <bean name="car"  class="com.itqf.spring.bean.Car" >
           <property name="name" value="mime"/>
           <property name="color" value="白色"/>
 </bean>
```

测试:

```
  @Test
  public  void test2(){
        //TODO 测试参构造方法
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext-injection.xml");

        Person person = context.getBean("person", Person.class);

        System.out.println(person);
        //结果:调用有参数构造方法,输出
  }
```

2.2. index属性：按参数索引注入

参数名一致，但位置不一致时，使用 `index`

例如以下两个构造函数（第二个是新添加）：

QFEDU 千锋教育 WWW.QFEDU.COM

```
    public Person(String name, Car car) {
        super();
        System.out.println("Person(String name, Car car)");
        this.name = name;
        this.car = car;
    }

    public Person(Car car, String name) {
        super();
        System.out.println("Person(Car car, String name)");
        this.name = name;
        this.car = car;
    }
```

配置：使用 `index` 确定调用哪个构造函数

```
    <bean name="person2" class="com.itqf.spring.bean.Person">
        <constructor-arg name="name" value="helen" index="0"></constructor-arg>
        <constructor-arg name="car" ref="car" index="1"></constructor-arg>
    </bean>
```

测试：

重新执行第一步的测试用例，执行结果调用了第一个构造函数

2.3. type属性：按参数类型注入

参数名和位置一致，但类型不一致时，使用`type`

例如以下两个构造函数（第二个是新添加）：

```
    public Person(Car car, String name) {
        super();
        System.out.println("Person(Car car, String name)");
        this.name = name;
        this.car = car;
    }

    public Person(Car car, Integer name) {
        super();
        System.out.println("Person(Car car, Integer name)");
        this.name = name + "";
        this.car = car;
    }
```

配置：使用`type`指定参数的类型

QFEDU 千锋教育 WWW.QFEDU.COM

```
    <bean name="person2" class="com.itqf.spring.bean.Person">
        <constructor-arg name="name" value="988" type="java.lang.Integer">
</constructor-arg>
        <constructor-arg name="car" ref="car" ></constructor-arg>
    </bean>
```

测试：

重新执行前面的测试用例，执行结果调用了第二个构造函数

3. p名称空间注入

导入p名称空间:

使用p:属性名 完成注入,走set方法

- 基本类型值: p:属性名="值"
- 引入类型值: P:属性名-ref="bean名称"

配置:

```
//1.第一步配置文件中 添加命名空间p
 xmlns:p="http://www.springframework.org/schema/p"

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

       //使用 p命称空间进行赋值
        <bean name="person" class="com.itqf.spring.bean.Person"  p:name="人名"
p:age="11"    p:car-ref="car">

       </bean>

       <bean name="car"  class="com.itqf.spring.bean.Car" >
           <property name="name" value="mime" />
           <property name="color" value="白色"/>
       </bean>
```

测试:

QFEDU 千锋教育 WWW.QFEDU.COM

```
@Test
public  void test2(){
    //TODO 测试p命名空间注入
    ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext-injection.xml");

    Person person = context.getBean("person", Person.class);

    System.out.println(person);

 }
```

4. spel注入

spring Expression Language：spring表达式语言

配置:

```
  <bean name="car"  class="com.itqf.spring.bean.Car" >
          <property name="name" value="mime" />
          <property name="color" value="白色"/>
  </bean>

   <!--利用spel引入car的属性 -->
   <bean  name="person1"  class="com.itqf.spring.bean.Person"  p:car-ref="car">
           <property name="name" value="#{car.name}"/>
           <property name="age" value="#{person.age}"/>

   </bean>
```

测试

```
@Test
public  void test3(){
    //TODO 测试spel注入
    ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext-injection.xml");

    Person person = context.getBean("person1", Person.class);

    System.out.println(person);

 }
```

QFEDU 千锋教育 WWW.QFEDU.COM

5. 复杂类型注入

创建配置文件:application-collection.xml

创建测试代码:CollectionTest.java

创建测试实体类:TestCollection

创建TestCollection:

```
/**
 * 练习:arr list map properties的注入
 */
public class TestCollection {

    private Object [] arrs;
    private List<Object> list;
    private Map<String,Object> map;
    private Properties properties;

    public Object[] getArrs() {
        return arrs;
    }

    public void setArrs(Object[] arrs) {
        this.arrs = arrs;
    }

    public List<Object> getList() {
        return list;
    }

    public void setList(List<Object> list) {
        this.list = list;
    }

    public Map<String, Object> getMap() {
        return map;
    }

    public void setMap(Map<String, Object> map) {
        this.map = map;
    }

    public Properties getProperties() {
        return properties;
    }

    public void setProperties(Properties properties) {
```

QFEDU 千锋教育 WWW.QFEDU.COM

```
        this.properties = properties;
    }

    @Override
    public String toString() {
        return "TestCollection{" +
                "arrs=" + Arrays.toString(arrs) +
                ", list=" + list +
                ", map=" + map +
                ", properties=" + properties +
                '}';
    }
}
```

配置:

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


       <bean name="car" class="com.itqf.spring.bean.Car">
             <property name="name" value="保时捷"/>
             <property name="color" value="红色" />
       </bean>


       <bean name="testColl" class="com.itqf.spring.bean.TestCollection">

           <!-- 数组变量注入 -->
           <property name="arrs">
               <list>
                   <value>数组1</value>
                   <!--引入其他类型-->
                   <ref bean="car"/>
               </list>
           </property>

           <!-- 集合变量赋值-->
           <property name="list">
                <list>
                    <value>集合1</value>
                    <!--集合变量内部包含集合-->
                    <list>
                        <value>集合中的集合1</value>
```

QFEDU 千锋教育 WWW.QFEDU.COM

```
                    <value>集合中的集合2</value>
                    <value>集合中的集合3</value>
                </list>
                <ref bean="car" />
            </list>
        </property>

        <!--map赋值 -->
        <property name="map">
            <map>
                <entry key="car" value-ref="car" />
                <entry key="name" value="保时捷" />
                <entry key="age"  value="11"/>
            </map>

        </property>

        <!-- properties赋值 -->
        <property name="properties">
            <props>
                <prop key="name">pro1</prop>
                <prop key="age">111</prop>
            </props>
        </property>
    </bean>

</beans>
```

测试:

```
 @Test
 public  void  test4(){
        //TODO 复杂类型注入练习
     ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext-collection.xml");

     TestCollection textColl = context.getBean("testColl", TestCollection.class);

     System.out.println("testColl = " + textColl);

   }
```

二.使用注解

> 使用注解的方式完成IOC

QFEDU 千锋教育 WWW.QFEDU.COM

1. 准备工作

- 创建项目: spring-02-annotation
- 导入jar包: spring-core,spring-context,spring-suppot-context,spring-beans,spring-expression, log4j,commons-logging,本次多加一个:spring-aop
- 引入日志配置文件:log4j.properties
- 实体类: 原项目 Person.java 和 Car.java即可
- 创建配置文件: applicationContext.xml
- 创建测试代码类:AnnotationTest.java

2. 使用注解

2.1 引入Context的约束

参考文件位置:spring-framework-4.2.4.RELEASE\docs\spring-framework-reference\html\xsd-configuration.html `40.2.8 the context schema`

配置:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--较之前多了 xmlns:context -->
<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
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->


</beans>
```

2.2 配置注解扫描

在applicationContext.xml中配置:

指定扫描 `包` 下所有类中的注解,扫描包时,会扫描包所有的子孙包.

QFEDU 千锋教育 WWW.QFEDU.COM

```
<?xml version="1.0" encoding="UTF-8"?>
<!--较之前多了 xmlns:context -->
<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
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->

     <!--扫描包设置-->
      <context:component-scan base-package="com.itqf.spring.bean">
</context:component-scan>
</beans>
```

2.3 使用注解

在Person类的头部添加如下注解

```
/**
  * @Component(person) == <bean name="person" class="com.itqf.spring.bean.Person"
/>
  */

@Component("person")
public class Person {
    private String name;
    private Integer age;
    private Car car;


    public  Person(){
        System.out.println("无参数构造方法!");
    }

  //getter,setter,toString
}
```

测试:

QFEDU 千锋教育 WWW.QFEDU.COM

```
  @Test
  public  void  test1(){
     //TODO 测试注解入门
     ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

     Person person = context.getBean("person", Person.class);

     System.out.println("person = " + person);
 }
```

3. 其他注解

介绍其他常用注解,测试方式同前

3.1 类头部可用的注解

```
@Service("person")      // service层
@Controller("person")   // controller层
@Repository("person")   // dao层
```

3.2 类头部可用的注解

指定对象作用域

```
@Scope(scopeName="singleton")
@Scope(scopeName="prototype")
```

3.3 注入属性value值

1.设置成员变量上:通过反射给变量赋值

```
@Value("name值")
private String name;
```

@Value("name值") 等同于 @Value(value="name值")

2.加在set方法上:通过set方法赋值

QFEDU 千锋教育 WWW.QFEDU.COM

```
@Value("tom")
public void setName(String name)
{
  this.name = name;
}
```

3.4 自动装配

1. @Autowired

使用 `@Autowired` 自动装配对象类型的属性: 下面的Person中的Car使用了自动装配

```
//将Car定义成接口
@Component
public interface Car {
     void log();
}

//Baoma实现Car
@Component
public class Baoma implements Car {

    public void log() {
        System.out.println("宝马");
    }
}

//XianDai实现Car
@Component
public class XianDai implements Car {

    public void log() {
        System.out.println("现代");
    }
}
```

装配类:

QFEDU 千锋教育 WWW.QFEDU.COM

```
@Scope(scopeName = "prototype")
@Component("person")
public class Person {
    @Value("name值")
    private String name;
    private Integer age;
    @Autowired
    private Car car;  //自动装配 可以选择Car,如果Car是接口,找Car的实现类!
```

注意: 以上操作会出现一个问题,如果Car是接口,且Car只有一个实现类,那么@Autowired会自动将实现类装配给Person的car变量上,但是如果Car是接口,并且有两个以上实现类,那么自动装配就会报错,无法选择由哪个实现类赋值.所以需要配合另一个注释@Qualifier("bean name"), 这个属性可以将@Autowired按类型赋值改成按bean名字赋值.

3.5 @Qualifier

- 如果匹配多个类型一致的对象，将无法选择具体注入哪一个对象
- 使用`@Qualifier()`注解告诉spring容器自动装配哪个名称的对象。

```
@Scope(scopeName = "prototype")
@Component("person")
public class Person {
    @Value("name值")
    private String name;
    private Integer age;
    @Autowired
    @Qualifier("baoma")   //指定实现类
    private Car car;  //自动装配 可以选择Car,如果Car是接口,找Car的实现类!
```

3.6 @Resource

@Resource 是java的注释,但是Spring框架支持,@Resource指定注入哪个名称的对象

@Resource("name") == @Autowired + @Qualifier("name")

```
@Resource("baoma")
private Car car;
```

3.7 初始化和销毁方法

初始化和销毁方法等同于配置文件添加的init-method和destroy-method功能,

例:Person类中init方法和destroy方法添加如下注解:

QFEDU 千锋教育 WWW.QFEDU.COM

```
@PostConstruct
public  void init(){
  System.out.println("初始化方法");
}

@PreDestroy
public void destroy(){
  System.out.println("销毁方法");
}
```

三. Spring整合JUnit测试

spring整合junit，为我们提供了方便的测试方式

1、导包：在spring-02-annotation项目中再加入如下包

spring-test-4.2.8.jar

2、创建测试类

```
//创建容器
@RunWith(SpringJUnit4ClassRunner.class)
//指定创建容器时使用哪个配置文件
@ContextConfiguration("classpath:applicationContext.xml")
public class RunWithTest {
    //将名为user的对象注入到u变量中
    @Resource(name="person")
    private Person p;
    @Test
    public void testCreatePerson(){
        System.out.println(p);
    }
}
```

课前默写

```
1. Spring的基本配置
2. 代码实现Spring创建对象过程
```

作业

QFEDU 千锋教育
WWW.QFEDU.COM

1. 创建汽车类（Car）和引擎接口（Engine）
2. 创建两个引擎的实现类V6Engine和V8Engine
3. 使用依赖注入的方式显示V6的汽车和V8的汽车奔跑（running）的效果
4. 使用BeanFactory和ApplicationContext分别加载以上的类
5. 使用不同的生命周期创建上面的类并使用JUnit测试不同之处
6. 使用注解的方式再实现一次

面试题

1. 在Spring中如何注入基本数据类型
2. @Component注解的作用
3. @Resource注解和@Autowired注解的作用和区别
4. Spring如何集成JUnit