Spring框架第一天

第一章:创建对象与依赖注入

1. Spring中基于XML的IOC环境搭建

```
1. 创建一个Spring的配置文件,名为bean.xml
2. 在配置文件中引入最基本的约束


        <?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 id = "ServiceDaoImpl" class="com.dao.ServiceDaoImpl"></bean>

            <bean id="ServiceImpl" class="com.service.ServiceImpl"></bean>

        </beans>

3.  在配置文件中使用bean标签来管理对象
4. 代码使用Spring容器示例

          ApplicationContext applicationContext =  new ClassPathXmlApplicationContext("bean.xml");
          IServiceDao iServiceDao = (IServiceDao) applicationContext.getBean("ServiceDaoImpl");
          IService service = (IService)applicationContext.getBean("ServiceImpl") ;
          System.out.println(iServiceDao);
          System.out.println(service);
```

2. 在spring配置文件中创建对象的三种方式

1. 使用默认构造函数创建对象

   - 待创建的java类

     ```
     public class ServiceImpl implements IService {
         public void save() {
             System.out.println("方法被执行了");
         }
     }
     ```

   - 在spring配置文件中的内容,使用默认构造函数创建出ServiceImpl对象

     ```
     <bean id="ServiceImpl" class="service.ServiceImpl" scope="singleton"></bean>
     ```

2. 使用某个类中的方法创建对象

   - 创建对象的类

     public class methodFactory {

     ```
     public IService getService(){
         return new ServiceImpl();
     }
     }
     ```

   - 在配置文件中,使用getService方法创建出ServiceImpl对象

     ```
     - 先创建出methodFactory这个对象
     ```

```
<bean id = "methodFactory" class="Factory.methodFactory"></bean>
- 再使用methodFctory对象中的getServic方法创建出ServiceImpl对象
<bean id = "methodbean" factory-bean="methodFactory" factory-method="getServic"></bean>
```

3. 使用某个类的静态方法创建对象
    - 静态工厂类

```
public class staticFactory {

    public static IService getServic(){
        return new ServiceImpl();
    }
}
```

    - 在配置文件中使用该静态方法创建对象

```
<bean id = "staticFactory" class="Factory.staticFactory" factory-method="getServic"></bean>
```

3. bean的作用范围和生命周期

1. 对象的作用范围:

```
1. singleton:单例的
2. prototype:多例的
3. request:web应用下的请求对象
4. session:web应用下的session范围
5. global-session:作用于集群环境下的session
```

2. bean的生命周期:

```
1. 单例对象:
   出生:容器创建时对象自动创建
   活着：容器活着它就活着
   销毁：容器销毁时便销毁
   容器中的对象与容器同生共死

2. 多例对象
   出生:当使用到对象时对象就会被创建
   活着：使用过程中就活着
   销毁：由垃圾回收器对其进行统一回收
```

4. 依赖注入:在对象属性中注入对应的值

1. 使用构造函数
    - java类

```
public class ServiceImpl implements IService {
    private int age;
    private String name;
    private Date birthday;

    public ServiceImpl(int age, String name, Date birthday) {
        this.age = age;
        this.name = name;
        this.birthday = birthday;
    }
}
```

    - 配置文件中的内容

```
 <bean id = "ServiceImpl" class="com.service.ServiceImpl" >
```

```
            <constructor-arg name="name" value="李四"></constructor-arg>
            <constructor-arg name="age" value="20"></constructor-arg>
            <constructor-arg name="birthday" ref = "birthday"></constructor-arg>
 </bean>
 <bean id = "birthday" class="java.util.Date"></bean>
name:指向对象中待注入数据的属性名
value：普通类型和String类型
ref:指向容器中其他的对象
```

2. 使用setter方法

   - 对应的java类

```
public class ServiceImpl implements IService {
        private int age;
        private String name;
        private Date birthday;

        public int getAge() {
            return age;
        }

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

        public String getName() {
            return name;
        }

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

        public Date getBirthday() {
            return birthday;
        }

        public void setBirthday(Date birthday) {
            this.birthday = birthday;
        }

    }
```

   - 配置文件中的配置

```
<bean id = "ServiceImpl2" class="com.service.ServiceImpl2" >
            <property name="age" value="22"></property>
            <property name="birthday" ref="birthday"></property>
            <property name="name" value="李四"></property>
</bean>
<bean id = "birthday" class="java.util.Date"></bean>
```

   - 注意

```
1.底层使用反射,所以直接通过配置文件中属性的名字找到对应的setter方法,所以name属性的值与
  对象的属性名并没有必然的关系,而与对象中的setter方法相关
2.举个例子来说,设置year属性的setter方法名为setYear,那么此时标签property属性name的值为
  year,但是假如setter方法名为setMyYear,那么此时property属性的name的值就变为myYear
```