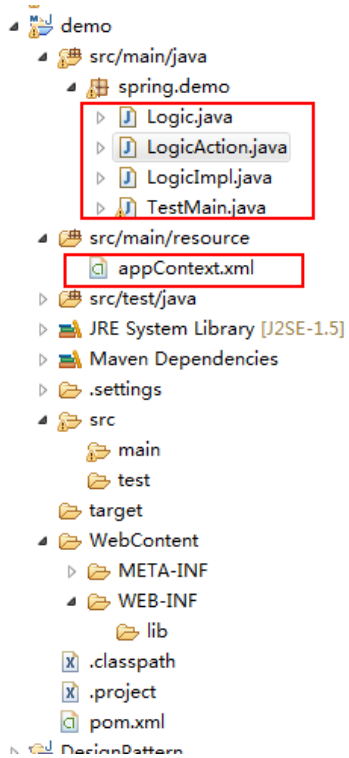
IOC实例一：


demo
src/main/java
spring.demo
Logic.java
LogicAction.java
LogicImpl.java
TestMain.java
src/main/resource
appContext.xml
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
.settings
src
main
test
target
WebContent
META-INF
WEB-INF
lib
.classpath
.project
pom.xml

LogicAction.java

```java
package spring.demo;


public class LogicAction {

    private Logic logic;

    public Logic getLogic() {
        return logic;
    }

    public void setLogic(Logic logic) {
        this.logic = logic;
    }

    public void execute() {
        String name = logic.getName();
        System.out.println("My name is " + name);
    }

}
```

FileService.... applicationC... appContext.xml Logic.java LogicImpl.java TestMain.java

```java
package spring.demo;

public class LogicImpl implements Logic{

    @Override
    public String getName() {
        return "shenhong";
    }

}
```

TestMain.java

```java
package spring.demo;

import org.springframework.context.ApplicationContext;

public class TestMain {

    public static  void main(String [] args) {
        System.out.println("ok");
        ApplicationContext ac = new ClassPathXmlApplicationContext("appContext.xml");
        LogicAction la = (LogicAction) ac.getBean("LogicAction");
        la.execute();
    }

}
```

LogicImpl.java

```java
package spring.demo;

public class LogicImpl implements Logic{

    @Override
    public String getName() {
        return "shenhong";
    }

}
```

appContext.xml

```xml
<?xml version="1.0"?>
<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-2.5.xsd">
    <!--配置spring的bean-->
    <bean id="LogicImpl"   class="spring.demo.LogicImpl"></bean>
    <bean id="LogicAction" class="spring.demo.LogicAction">
        <property name="logic"  ref="LogicImpl"></property>
    </bean>
</beans>
```

向IOC容器中注入对象，通过配置xml文件的<bean>节点来实现，<bean>中最主要的属性有两个，id和class，id表示标识这个<bean>节点，class表示关联的类文件名称（包名+类名）。<property>节点可以调用类中的setter方法，name对应参数名称，value对应传入的参数值。<constructor-arg>节点可以调用类中的构造器，name对应参数名称，value对应参数值。

Spring向IOC容器注入对象的四种方式：

1.利用无参构造函数+setter方法注入值（上面的例子就是无参+setter），对于LogicAction来说必须有无参构造函数，要不然会报错。

2. 利用有参构造函数直接注入。

```
public class Person {

    private String name;
    private Integer id;

    public Person(String name, Integer id) {
        super();
        this.name = name;
        this.id = id;
    }
```

```
<?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 class="com.mc.base.learn.spring.bean.Person" id="person">
        <constructor-arg name="id" value="123"></constructor-arg>
        <constructor-arg name="name" value="LiuChunfu"></constructor-arg>
    </bean>

</beans>
```

通过constructor-arg的方式来注入。

3. 通过静态工厂方法来注入

```
    public static Person createPerson(){
        return new Person();
    }

    /**
     * 工厂方法带有参数如何处理？
     * @Title: createPerson
     * @Description: TODO(这里用一句话描述这个方法的作用)
     * @param  @param id
     * @param  @param name
     * @param  @return
     * @return Person    返回类型
     * @throws
     */
    public static Person createPerson(Integer id,String name){
        return new Person(name,id);
    }
}
```

```
    <!--静态的工厂方法核心是class+factory-method -->
     <bean id="person" class="com.mc.base.learn.spring.factory.PersonStaticFactory" factory-method="createPerson">
        <!--通过property方法向createPerson传递参数 -->
        <property name="name" value="LiuChunfu"></property>
        <property name="id" value="125"></property>
    </bean>
```

测试如下：

```
    @Test
    public void testName() throws Exception {
        ApplicationContext ac=new ClassPathXmlApplicationContext("applicationContext.xml");
        Person person=ac.getBean("person3", Person.class);
        System.out.println(person);//Person [name=LiuChunfu, id=125]
    }
```

四，通过实例工厂方式创建对象

三、通过实例工厂方式创建对象。

实例工厂，就是通过实例来调用对象，但是所得到的对象最终也是单利模式。实例工厂和静态工厂创建的对象都是单例模式，两者的区别就是创建对象的实际不同，静态工厂是在创建容器的时候就创建了，实例工厂是在调用方法的时候才创建。知道Java设计模式中的单例模式设计（饿汉式和懒汉式）的读者，对这里的静态工厂模式和实例工厂模式肯定有所体会。

```
package com.mc.base.learn.spring.factory;

import com.mc.base.learn.spring.bean.Person;

public class PersonFactory {

    public Person createInstance() {
        return new Person();
    }
}
```

```
    <bean id="personFactory" class="cn.test.util.PresonFactoryInstance"></bean>
    <bean id="person4" factory-bean="personFactory" factory-method="createPerson">
        <property name="name" value="LiuChunfu"></property>
        <property name="id" value="125"></property>
    </bean>
```

```
    @Test
    public void testName() throws Exception {
        ApplicationContext ac=new ClassPathXmlApplicationContext("applicationContext.xml");
        Person person=ac.getBean("person4",Person.class);
        System.out.println(person);//Person [name=LiuChunfu, id=125]
    }
```