1.1 案例需求

需求概述

1.2 相关知识点

1.2.1 Spring 概述

1.2.1.1 什么是 Spring

EE开发分成三层结构：

- WEB 层:Spring MVC.
- 业务层:Bean管理:(IOC)
- 持久层:Spring的JDBC模板.0RM模板用于整合其他的持久层框架.

Expert One-to-One J2EE Design and Development :J2EE 的设计和开发:(2002.EJB)

Expert One-to-One J2EE Development without EJB :J2EE 不使用 EJB 的开发.

1.2.1.2 为什么学习 Spring

- 方便解耦，简化开发

  Spring就是一个大工厂，可以将所有对象创建和依赖关系维护，交给Spring管理

- AOP编程的支持

  Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能

- 声明式事务的支持

  只需要通过配置就可以完成对事务的管理，而无需手动编程

- 方便程序的测试

  Spring对Junit4支持，可以通过注解方便的测试Spring程序

- 方便集成各种优秀框架

  Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架(如：Struts、Hibernate> MyBatis、Quartz等)的直接支持

- 降低JavaEEAPI的使用难度

  Spring对JavaEE开发中非常难用的一些API （JDBC、JavaMail、远程调用等），都提供了封装, 使这些API应用难度大大降低

1.2.1.3 Spring 的版本

Spring 3.X 和 Spring4.X

1.2.2 Spring 的入门案例:(IOC)

1.2.2.1 IOC的底层实现原理

- IOC： Inversion of Control
  - 控制反转.指的是 对象的创建权反转（交给）给Spring.作用是实现了程序的解耦合.

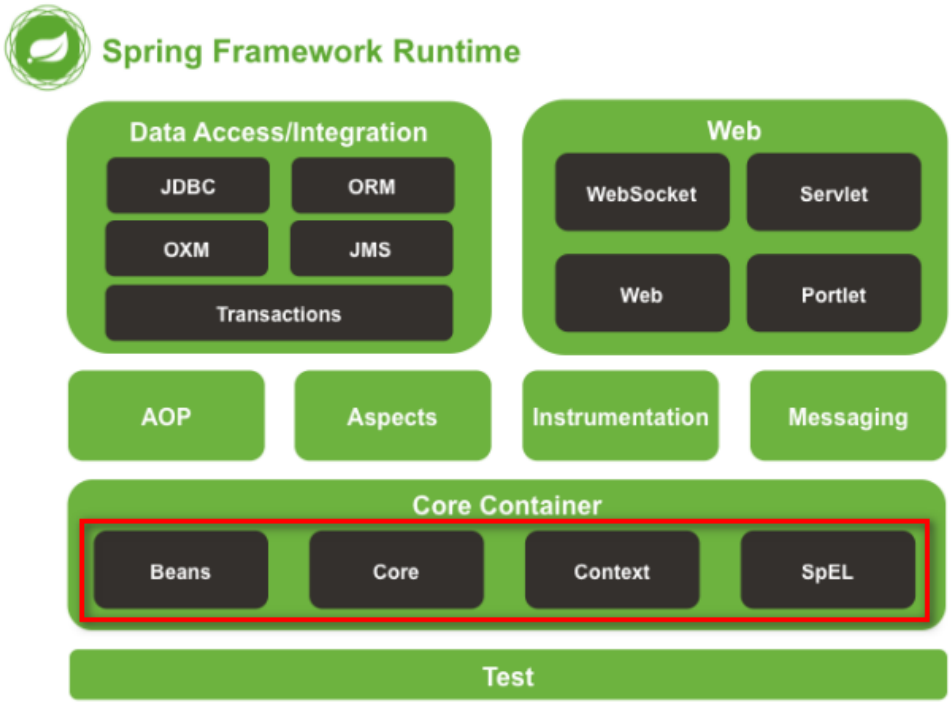
1.2.2.2 步骤一:下载Spring的开发包

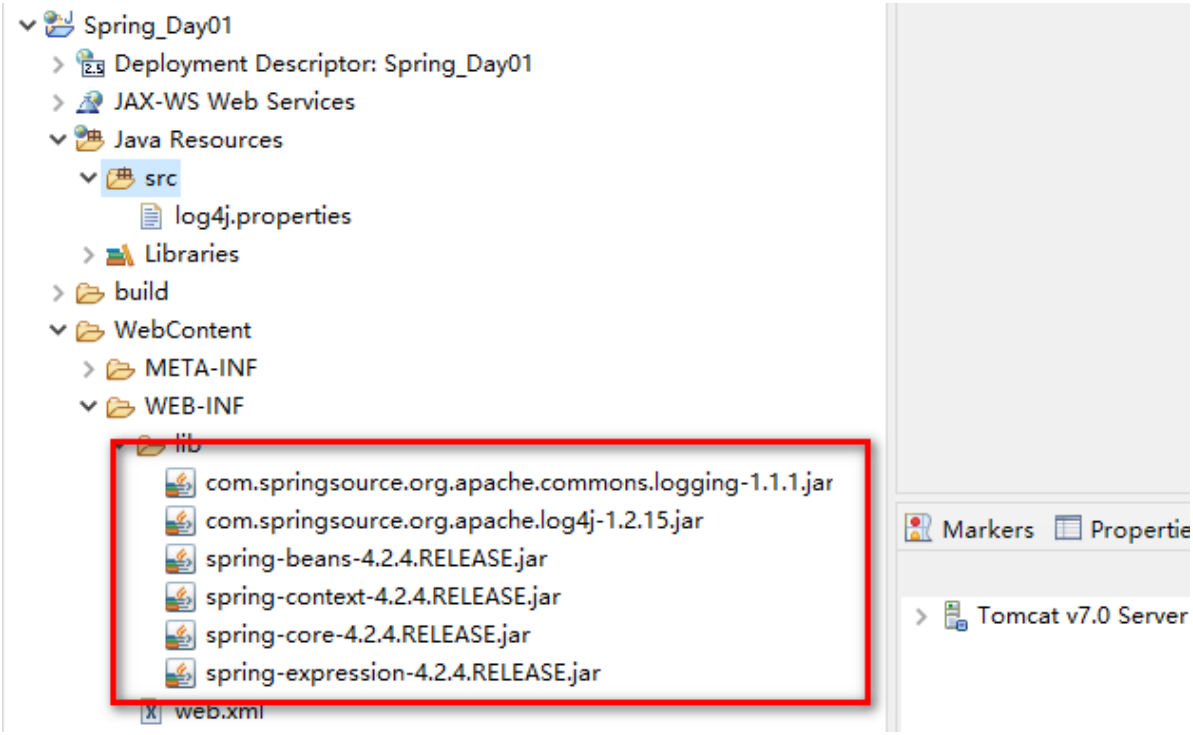
官网：http://spring.io/

名称	修改日期	类型	大小
docs	2015/12/17 0:59	文件夹	
libs	2015/12/17 0:59	文件夹	
schema	2015/12/17 0:59	文件夹	
license	2015/12/17 0:43	文本文档	15 KB
notice	2015/12/17 0:43	文本文档	1 KB
readme	2015/12/17 0:43	文本文档	1 KB

- docs ： Spring的开发规范和API
- libs ： Spring的开发的jar和源码
- schema ： Spring的配置文件的约束

1.2.2.3 骤二:创建web项目，引入Spring的开发包





1.2.2.4 骤三:引入相关配置文件

1.2.2.4 编写相关的类

UserDao.java

```
/**
* @Title: UserDao.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo1;

public interface UserDao {

    public void save();
}
```

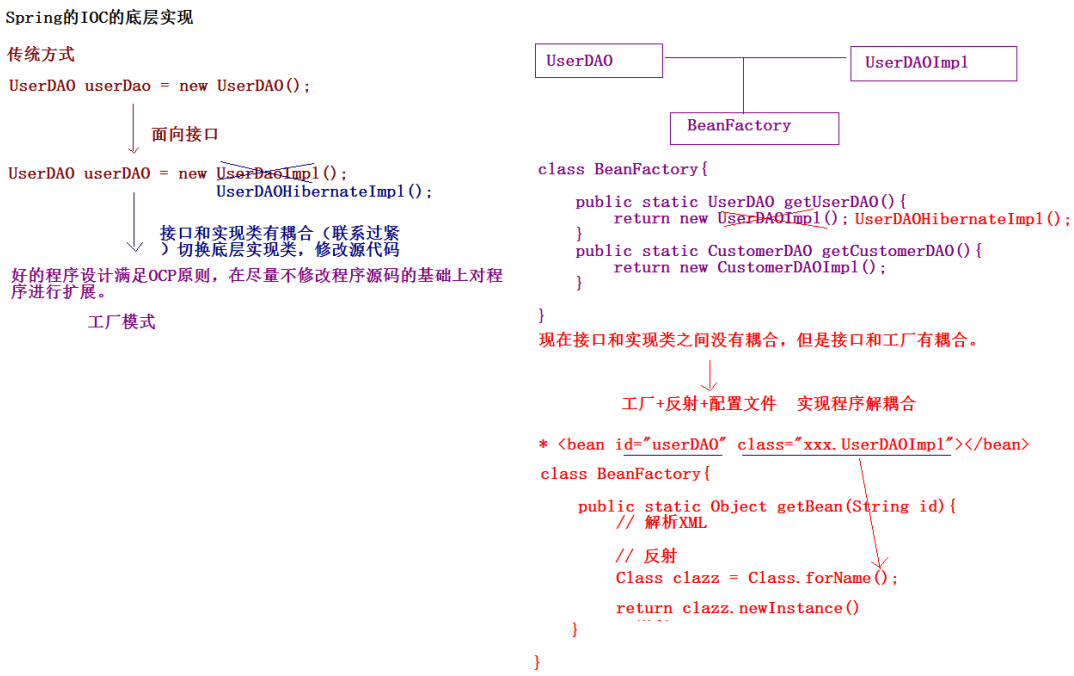
UserDaoImpl.java

```
/**
* @Title: UserDaoImpl.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo1;

public class UserDaoImpl implements UserDao {

    @Override
    public void save() {
        System.out.println("UserDaoImpl 执行了...");
    }

}
```

UserDaoHibernateImpl.java

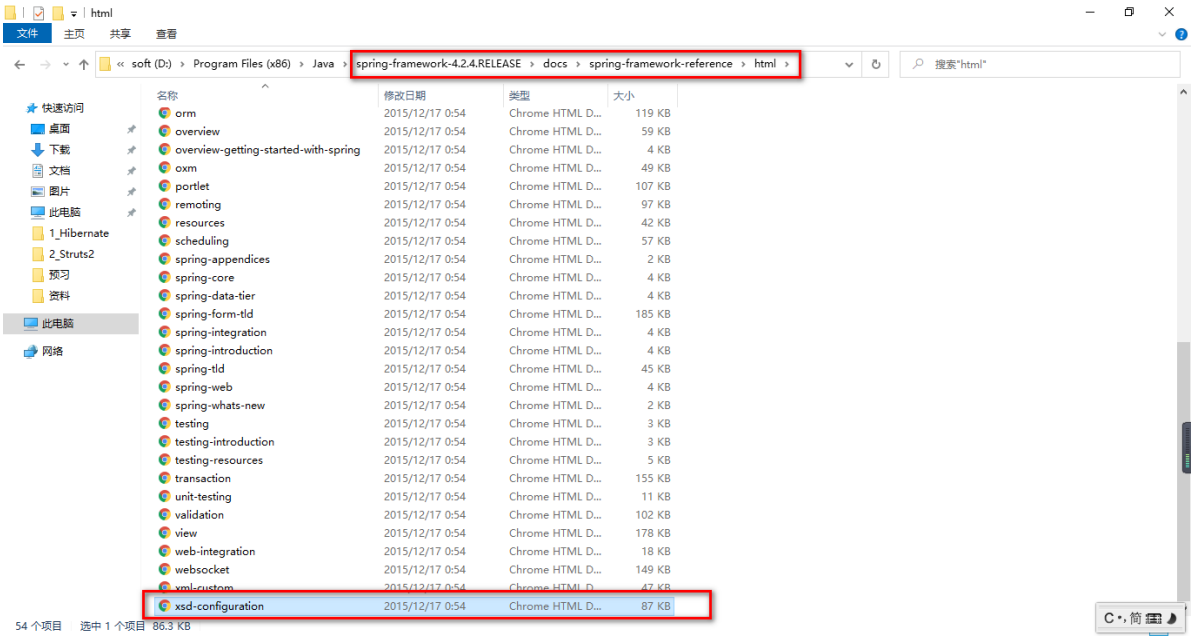
```
/**
* @Title: UserDaoHibernateImpl.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo1;
```

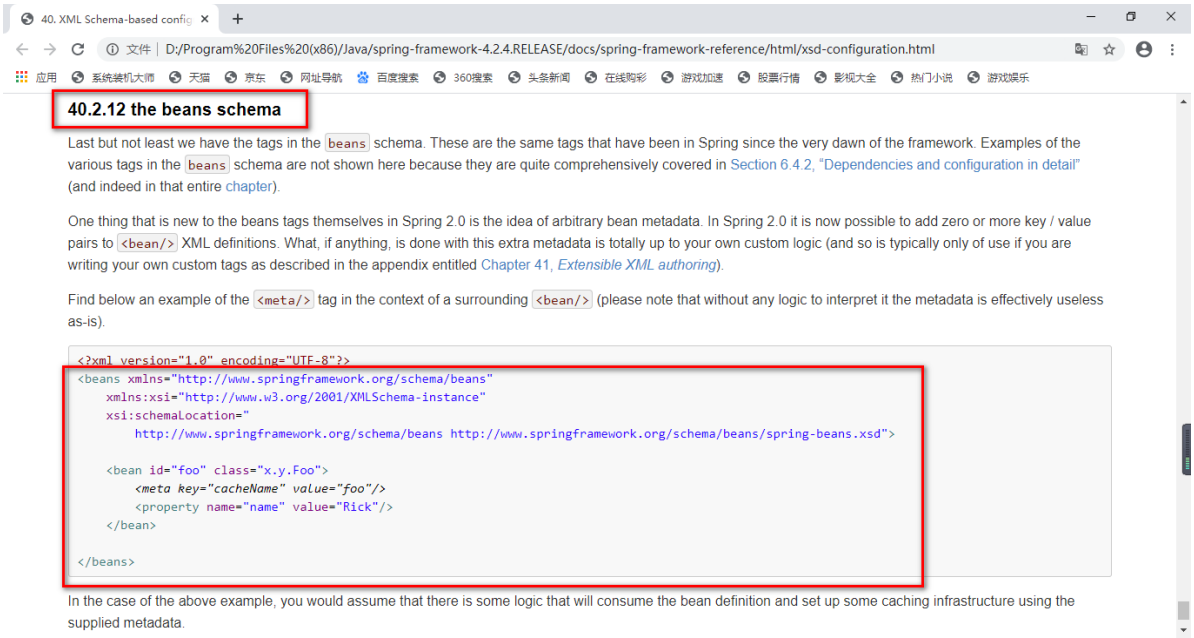
```java
public class UserDaoHibernateImpl implements UserDao {

    @Override
    public void save() {
        System.out.println("UserDaoHibernateImpl 执行了....");
    }

}
```



1.2.2.6 步骤五:完成配置

- 将实现类交给 Spring 管理





- 新建 applicationContext.xml

```
<?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="userDao" class="com.admiral.spring.demo1.UserDaoImpl"></bean>


</beans>
```

1.2.2.7 步骤六:编写测试程序

```
/**
* @Title: SpringDemo1.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo1;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo1 {

    @Test
    /**
     * 传统方式调用
     */
    public void demo1() {
        UserDao userDao = new UserDaoHibernateImpl();
        userDao.save();
    }

    @Test
    /**
     * Spring 的方式调用
     */
    public void demo2() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        UserDao userDao = (UserDao) applicationContext.getBean("userDao");
        userDao.save();
    }
}
```

1.2.2.8 IOC 和 DI:

- IOC：控制反转，将对象的创建权反转给了Spring。
- DI：依赖注入，前提必须有IOC的环境，Spring管理这个类的时候将类的依赖的属性注入（设置）进来。
- 面向对象的时候
    - 依赖

```
Class A{

}

Class B{

    public void xxx(A a){

    }
}
```

- 继承:is a

```
Class A{

}

Class B extends A{

}
```

- 聚合:has a

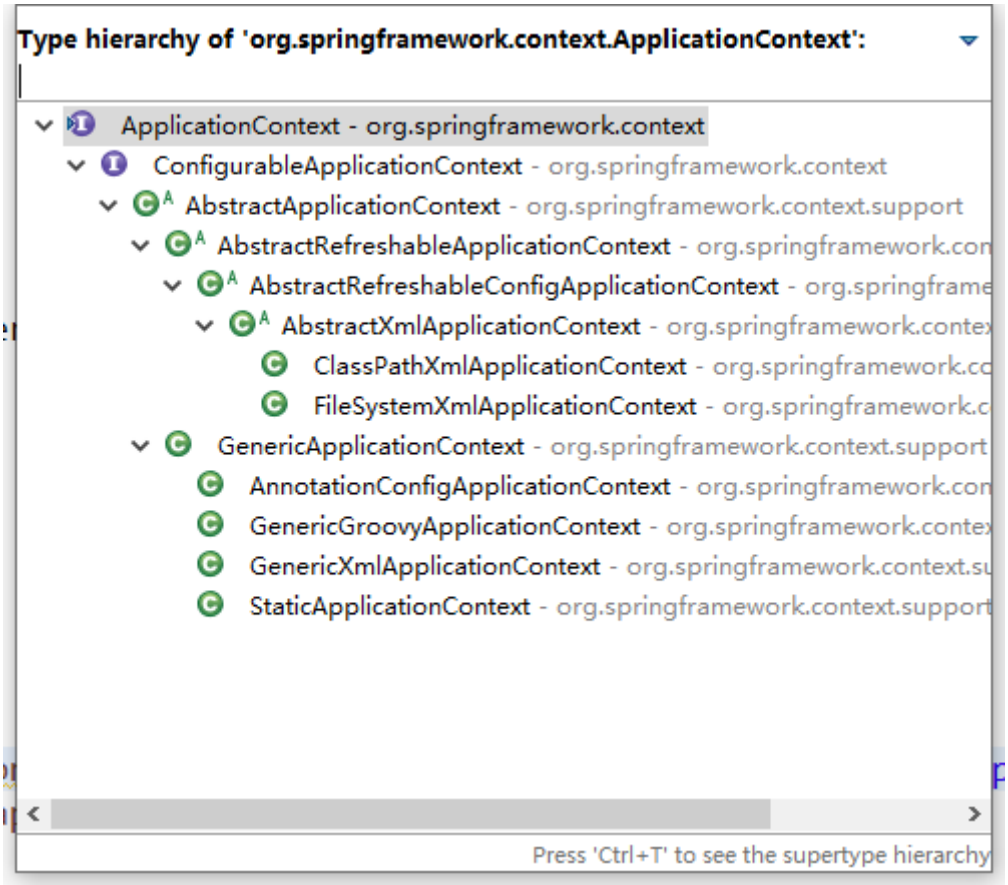
```
/**
* @Title: UserDaoImpl.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo1;

public class UserDaoImpl implements UserDao {

    private String name;

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

    @Override
    public void save() {
        System.out.println("UserDaoImpl 执行了..." + name);
    }

}
```

```xml
<?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="userDao" class="com.admiral.spring.demo1.UserDaoImpl">
        <property name="name" value="小红红"></property>
    </bean>


</beans>
```

1.2.3 Spring中的工厂

1.2.3.1 ApplicationContext:

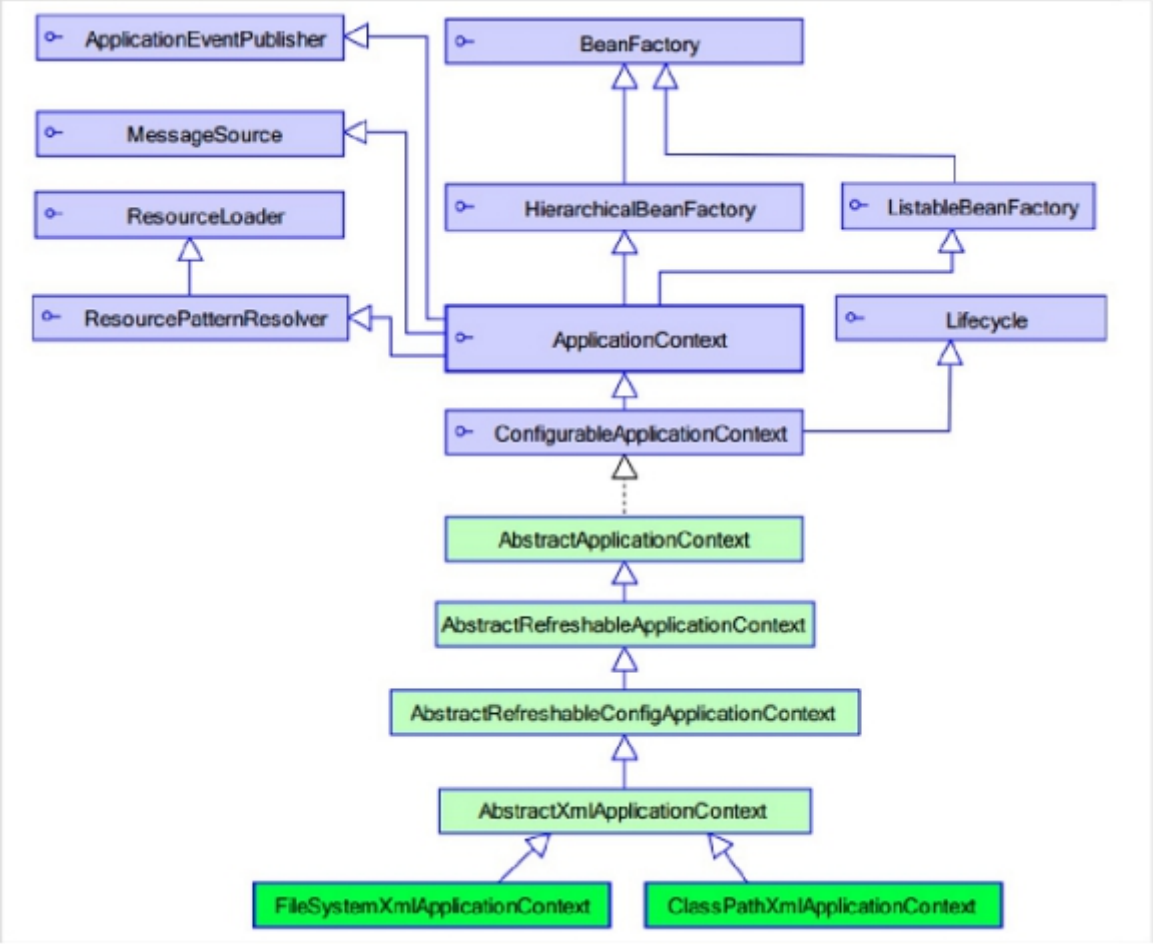
- ApplicatioContext接口有两个实现类:



> ClassPathXmlApplicationContext :加载类路径下Spring的配置文件.

> FileSystemXmlApplicationContext :加载本地磁盘下Spring的配置文件.

1.2.3.2 BeanFactory



1.2.3.3 BeanFactory 和 Applicationcontext 的区别

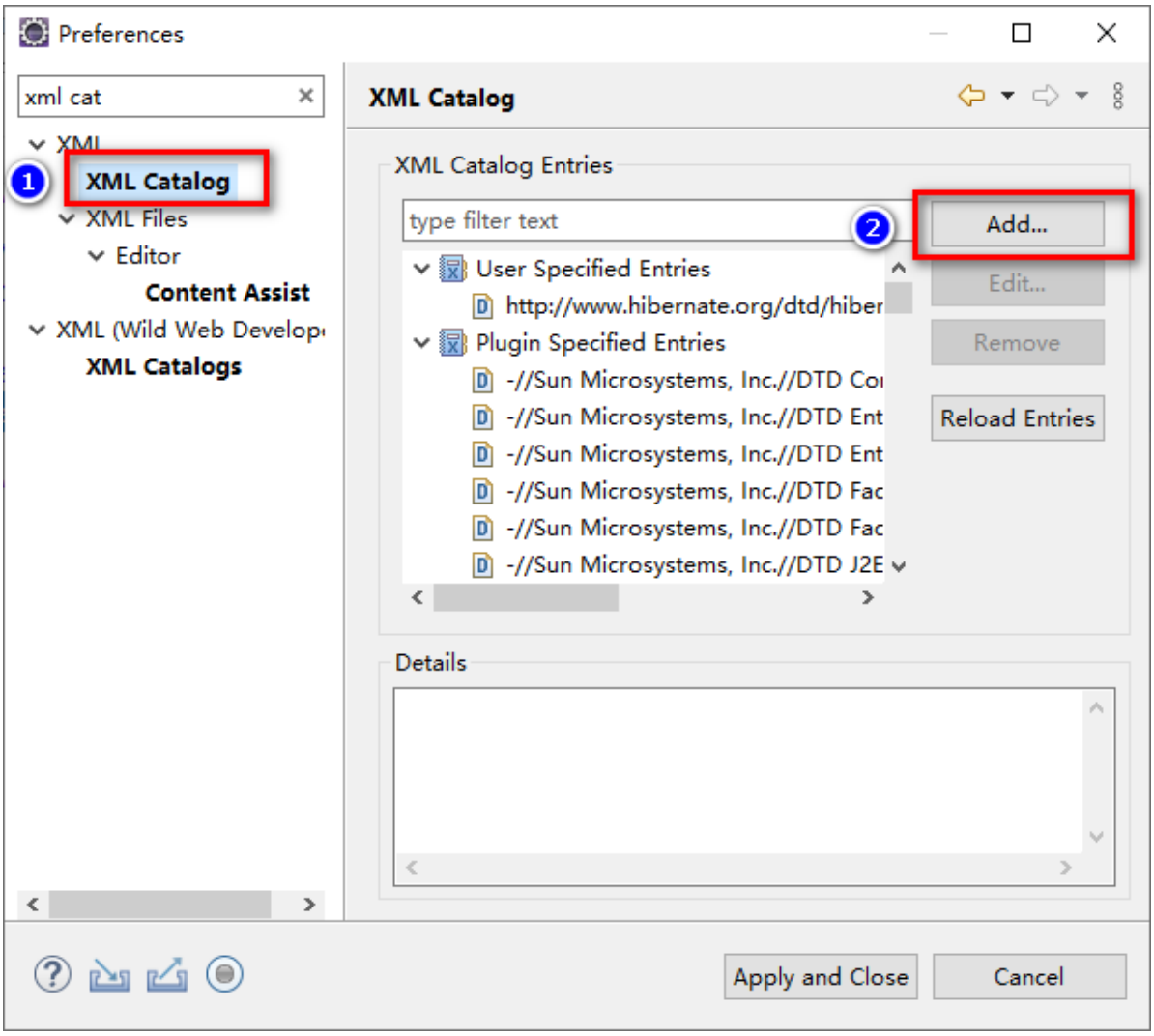
- BeanFactory :是在getBean的时候才会生成类的实例.
- Applicationcontext :在加载 applicationcontext.xml 时候就会创建.

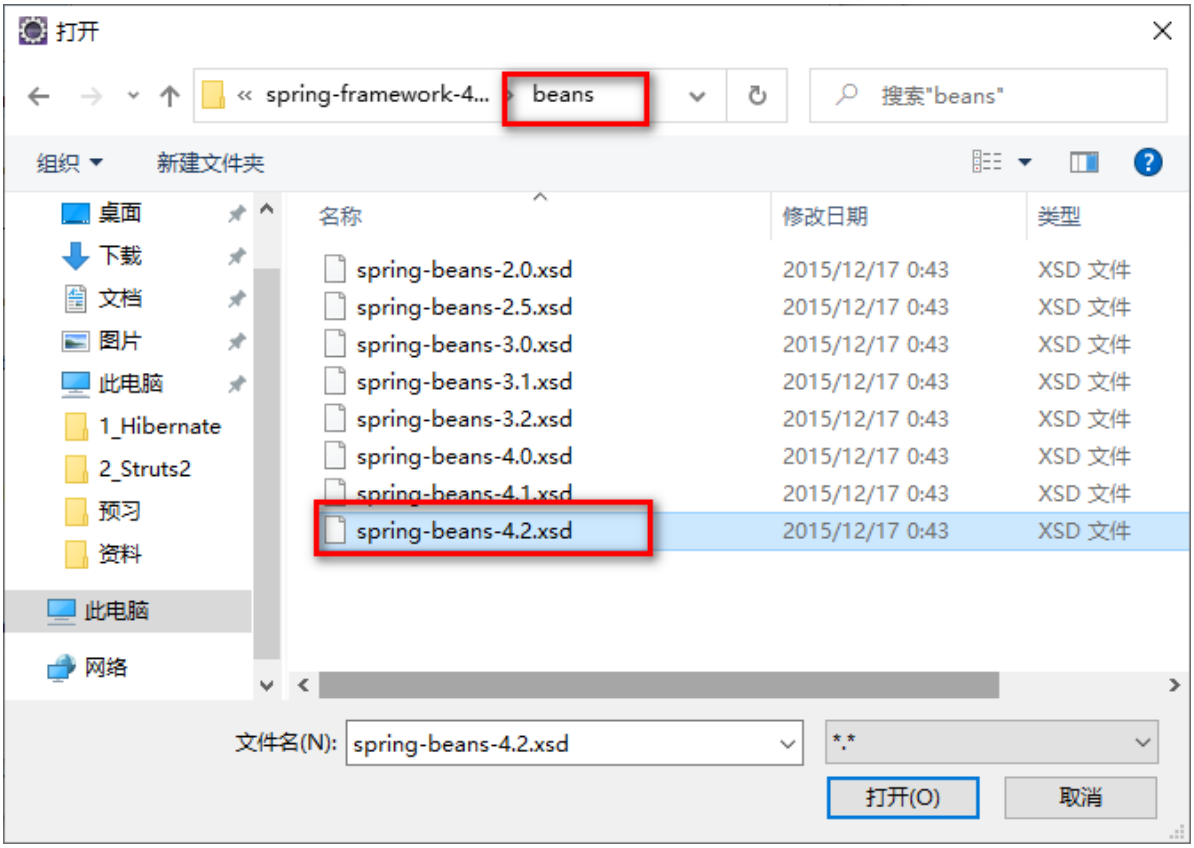
1.2.4 配置STS的XML的提示:

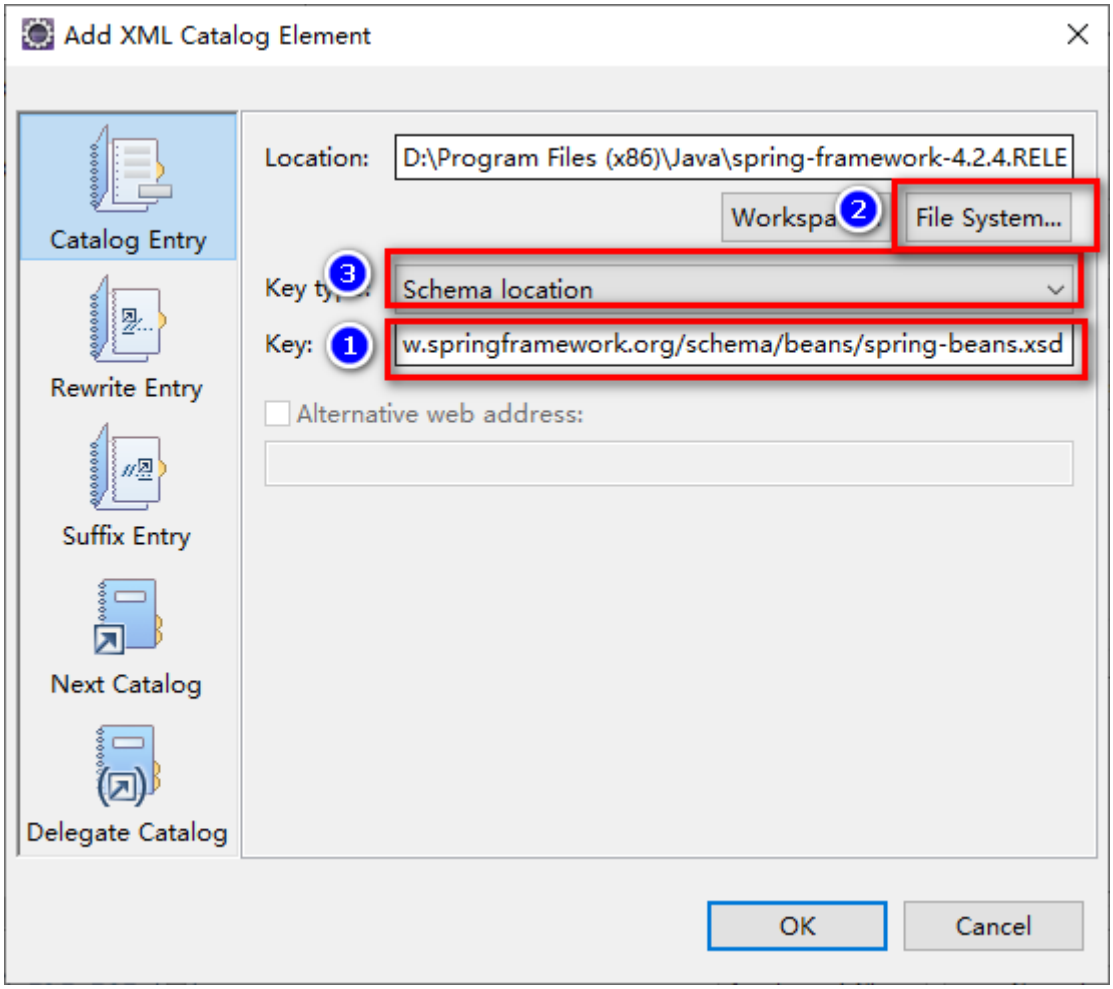
1.2.4.1 Spring配置文件中提示的配置

UserDao.java | UserDaoImpl.java | SpringDemo1.java | UserDaoHibernateImpl.java | *applicationContext.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="
5          http://www.springframework.org/schema/beans
6          http://www.springframework.org/schema/beans/spring-beans.xsd">
7
8      <bean id="userDao" class="com.admiral.spring.demo1.UserDaoImpl">
9          <property name="name" value="小红红"></property>
10     </bean>
11
12
13 </beans>
14
```


Preferences
xml cat
XML
1
XML Catalog
XML Files
Editor
Content Assist
XML (Wild Web Develop
XML Catalogs
XML Catalog
XML Catalog Entries
type filter text
2
Add...
Edit...
Remove
Reload Entries
User Specified Entries
http://www.hibernate.org/dtd/hiber
Plugin Specified Entries
-//Sun Microsystems, Inc.//DTD Co
-//Sun Microsystems, Inc.//DTD Ent
-//Sun Microsystems, Inc.//DTD Ent
-//Sun Microsystems, Inc.//DTD Fac
-//Sun Microsystems, Inc.//DTD Fac
-//Sun Microsystems, Inc.//DTD J2E
Details
Apply and Close
Cancel


打开
spring-framework-4...
beans
搜索"beans"
组织
新建文件夹
桌面
下载
文档
图片
此电脑
1_Hibernate
2_Struts2
预习
资料
此电脑
网络
名称
修改日期
类型
spring-beans-2.0.xsd 2015/12/17 0:43 XSD 文件
spring-beans-2.5.xsd 2015/12/17 0:43 XSD 文件
spring-beans-3.0.xsd 2015/12/17 0:43 XSD 文件
spring-beans-3.1.xsd 2015/12/17 0:43 XSD 文件
spring-beans-3.2.xsd 2015/12/17 0:43 XSD 文件
spring-beans-4.0.xsd 2015/12/17 0:43 XSD 文件
spring-beans-4.1.xsd 2015/12/17 0:43 XSD 文件
spring-beans-4.2.xsd 2015/12/17 0:43 XSD 文件
文件名(N): spring-beans-4.2.xsd
.
打开(O)
取消


Add XML Catalog Element
Catalog Entry
Rewrite Entry
Suffix Entry
Next Catalog
Delegate Catalog
Location:
D:\Program Files (x86)\Java\spring-framework-4.2.4.RELE
Workspa
File System...
Key t
Schema location
Key:
w.springframework.org/schema/beans/spring-beans.xsd
Alternative web address:
OK
Cancel

1.2.5 Spring的相关配置:

1.2.5.1 id属性和name属性标签的配置

- id　:使用了约束中的唯一约束。里面不能出现特殊字符的。
- name :没有使用约束中的唯一约束（理论上可以出现重复的，但是实际开发不能出现的）。里面可以出现特殊字符。
  - Spring和Struts1框架整合的时候
  - <bean name="/user" class=""/>

1.2.5.2 Bean的生命周期的配置(了解)

- init-method　　:Bean被初始化的时候执行的方法
- destroy-method　:Bean被销毁的时候执行的方法（Bean是单例创建，工厂关闭）

```
/**
* @Title: CustomerDaoImpl.java
* @Package com.admiral.spring.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo2;

public class CustomerDaoImpl implements CustomerDao {


    public void init() {
        System.out.println("CustomerDaoImpl 被实例化了....");
    }

    @Override
    public void save() {
        System.out.println("CustomerDaoImpl 执行了 ....");

    }

    public void destroy() {
        System.out.println("CustomerDaoImpl 销毁了 ....");
    }


}
```

```
<?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="customerDao" init-method="init"  destroy-method="destroy"
class="com.admiral.spring.demo2.CustomerDaoImpl"></bean>


</beans>
```

测试类:

```
/**
* @Title: SpringDemo2.java
* @Package com.admiral.spring.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo2;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo2 {

    @Test
    /**
     * init-method
     * destroy-method
     */
    public void demo1() {
        ClassPathXmlApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        CustomerDao customerDao = (CustomerDao)
applicationContext.getBean("customerDao");
        customerDao.save();
        applicationContext.close();
    }
}
```

1.2.5.3 scope属性:Bean的作用范围

- scope　　：Bean的作用范围
  - **singleton　：默认的，Spring会采用单例模式创建这个对象。**
  - **prototype　：多例模式。（）**
  - request　　：应用在web项目中，Spring创建这个类以后，将这个类存入到request范围中。
  - session　　：应用在web项目中，Spring创建这个类以后，将这个类存入到session范围中。
  - globalsession　：应用在web项目中，必须在porlet环境下使用。但是如果没有这种环境，相对于session。

```
<?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="userDao" class="com.admiral.spring.demo1.UserDaoImpl">
        <property name="name" value="小红红"></property>
    </bean>

    <bean id="customerDao" scope="prototype" init-method="init"  destroy-
method="destroy" class="com.admiral.spring.demo2.CustomerDaoImpl"></bean>


</beans>
```

```
/**
* @Title: SpringDemo2.java
* @Package com.admiral.spring.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo2;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo2 {



    @Test
    /**
     * bean 生命周期
     */
```

```
    public void demo2() {
        ClassPathXmlApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        CustomerDao customerDao1 = (CustomerDao)
applicationContext.getBean("customerDao");
        System.out.println(customerDao1);

        CustomerDao customerDao2 = (CustomerDao)
applicationContext.getBean("customerDao");
        System.out.println(customerDao2);

        System.out.println(customerDao1 == customerDao2);

        customerDao1.save();
        applicationContext.close();
    }
}
```

```
<terminated> SpringDemo2.demo2 [JUnit] D:\Program Files (x86)\Java\jre1.8.0_65\bin\javaw.exe  (2020-10-9 4:06:04 – 4:06:05)
04:06:05,147  INFO ClassPathXmlApplicationContext:578 - Refreshing org.springframework.context.suppor
04:06:05,198  INFO XmlBeanDefinitionReader:317 - Loading XML bean definitions from class path resourc
CustomerDaoImpl 被实例化了....
04:06:05,421  INFO ClassPathXmlApplicationContext:960 - Closing org.springframework.context.support.C
com.admiral.spring.demo2.CustomerDaoImpl@5123a213
CustomerDaoImpl 被实例化了....
com.admiral.spring.demo2.CustomerDaoImpl@52525845
false
CustomerDaoImpl 执行了 ....
```

1.2.6 Spring的Bean的管理XML的方式

1.2.6.1 Spring生成Bean的时候三种方式（了解）

1.2.6.2 Spring的Bean的属性注入

- 方式一:构造方法

```
/**
* @Title: Car.java
* @Package com.admiral.spring.demo3
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo3;

public class Car {
    private String name;
    private Double price;

    public Car(String name, Double price) {
```

```
        super();
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Car [name=" + name + ", price=" + price + "]";
    }

}
```

```
<?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">

    <!-- Spring Bean 属性的注入 -->
    <bean id="car" class="com.admiral.spring.demo3.Car">
        <constructor-arg name="name" value="宝马"/>
        <constructor-arg name="price" value="450000"/>
    </bean>


</beans>
```

```
/**
* @Title: SpringDemo3.java
* @Package com.admiral.spring.demo3
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo3;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo3 {

    @Test
    /**
     * bean 属性注入方式一:构造方法
     */
    public void demo1() {
```

```
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Car car = (Car) applicationContext.getBean("car");
        System.out.println(car);
    }
}
```

- 方式二:set方法

```
/**
* @Title: Car2.java
* @Package com.admiral.spring.demo3
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo3;

public class Car2 {

    private String name;
    private Double price;

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

    public void setPrice(Double price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Car2 [name=" + name + ", price=" + price + "]";
    }

}
```

```
<?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">
```

```
    <!-- Spring Bean 属性的注入 -->
    <bean id="car" class="com.admiral.spring.demo3.Car">
        <constructor-arg name="name" value="宝马"/>
        <constructor-arg name="price" value="450000"/>
    </bean>
    <bean id="car2" class="com.admiral.spring.demo3.Car2">
        <property name="name" value="奔驰" />
        <property name="price" value="1000000" />
    </bean>

</beans>
```

```
/**
* @Title: SpringDemo3.java
* @Package com.admiral.spring.demo3
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo3;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo3 {

    @Test
    /**
     * bean 属性注入方式一:构造方法
     */
    public void demo1() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Car car = (Car) applicationContext.getBean("car");
        System.out.println(car);
    }

    @Test
    /**
     * bean 属性注入方式一:set 方法
     */
    public void demo2() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Car2 car2 = (Car2) applicationContext.getBean("car2");
        System.out.println(car2);
    }
}
```

1.2.6.3 Spring的属性注入：对象类型的注入

```java
/**
* @Title: Person.java
* @Package com.admiral.spring.demo3
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo3;

public class Person {

    private String name;
    private Car2 car2;

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

    public void setCar2(Car2 car2) {
        this.car2 = car2;
    }

    @Override
    public String toString() {
        return "Person [name=" + name + ", car2=" + car2 + "]";
    }

}
```

```xml
<?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="userDao" class="com.admiral.spring.demo1.UserDaoImpl">
        <property name="name" value="小红红"></property>
    </bean>

    <bean id="customerDao" scope="prototype" init-method="init"
        destroy-method="destroy"
class="com.admiral.spring.demo2.CustomerDaoImpl">
    </bean>

    <!-- Spring Bean 属性的注入 -->
```

```
    <bean id="car" class="com.admiral.spring.demo3.Car">
        <constructor-arg name="name" value="宝马"/>
        <constructor-arg name="price" value="450000"/>
    </bean>

    <bean id="car2" class="com.admiral.spring.demo3.Car2">
        <property name="name" value="奔驰" />
        <property name="price" value="1000000" />
    </bean>

    <bean id="person" class="com.admiral.spring.demo3.Person">
        <property name="name" value="小花花" />
        <property name="car2" ref="car2" />
    </bean>

</beans>
```

```
/**
* @Title: SpringDemo3.java
* @Package com.admiral.spring.demo3
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo3;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo3 {

    @Test
    /**
     * bean 属性注入方式一:构造方法
     */
    public void demo1() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Car car = (Car) applicationContext.getBean("car");
        System.out.println(car);
    }

    @Test
    /**
     * bean 属性注入方式一:set 方法
     */
    public void demo2() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Car2 car2 = (Car2) applicationContext.getBean("car2");
```

```
        System.out.println(car2);
    }

    @Test
    /**
     * bean 对象类型的注入:set方法
     */
    public void demo3() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Person person = (Person) applicationContext.getBean("person");
        System.out.println(person);
    }
}
```

1.2.6.4 名称空间p的属性注入的方式:Spring2.x版本后提供的方式

- 通引过入p名称空间完成属性的注入：
  - 写法：
    - 普通属性 p:属性名="值"
    - 对象属性 p:属性名-ref="值"

- 引入 p 名称空间

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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="userDao" class="com.admiral.spring.demo1.UserDaoImpl">
```

- 使用 p 名称空间

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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="userDao" class="com.admiral.spring.demo1.UserDaoImpl">
        <property name="name" value="小红红"></property>
    </bean>

    <bean id="customerDao" scope="prototype" init-method="init"
        destroy-method="destroy"
        class="com.admiral.spring.demo2.CustomerDaoImpl">
```

```
    </bean>

    <!-- ============= Spring Bean 属性的注入 ============= -->
    <!-- 构造方法 -->
    <bean id="car" class="com.admiral.spring.demo3.Car">
        <constructor-arg name="name" value="宝马" />
        <constructor-arg name="price" value="450000" />
    </bean>

    <!-- set方法 -->
    <!-- <bean id="car2" class="com.admiral.spring.demo3.Car2"> <property
name="name"
        value="奔驰" /> <property name="price" value="1000000" /> </bean> -->

    <!-- set方法方式:注入对象类型 -->
<!--    <bean id="person" class="com.admiral.spring.demo3.Person">
        <property name="name" value="小花花" />
        <property name="car2" ref="car2" />
    </bean> -->

    <!-- ============= p 名称空间 ============= -->
    <bean id="car2" class="com.admiral.spring.demo3.Car2" p:name="夏利"
        p:price="20000"></bean>
    <bean id="person" class="com.admiral.spring.demo3.Person" p:name="小红红"
p:car2-ref="car2">
    </bean>




</beans>
```

1.2.6.5 SpEL的方式的属性注A：Spring3.x版本后提供的方式.

- SpEL：Spring Expression Language，Spring的表达式语言。
  - 语法：
    - #{SpEL}

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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">
```

```
    <!-- ============= SpEL ============= -->

    <bean id="car2" class="com.admiral.spring.demo3.Car2">
        <property name="name" value="#{'三蹦子'}" />
        <property name="price" value="#{3000}" />
    </bean>

    <bean id="person" class="com.admiral.spring.demo3.Person">
        <property name="name" value="#{'小黄黄'}" />
        <property name="car2" ref="car2" />
    </bean>


</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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">


    <!-- ============= SpEL ============= -->

    <bean id="carInfo" class="com.admiral.spring.demo3.CarInfo"></bean>


    <bean id="car2" class="com.admiral.spring.demo3.Car2">
        <property name="name" value="#{carInfo.name}" />
        <property name="price" value="#{carInfo.getPrice()}" />
    </bean>

    <bean id="person" class="com.admiral.spring.demo3.Person">
        <property name="name" value="#{'小黄黄'}" />
        <property name="car2" ref="car2" />
    </bean>



</beans>
```

1.2.6.6 注入复杂类型

```
/**
* @Title: CollectionBean.java
* @Package com.admiral.spring.demo4
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.spring.demo4;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class CollectionBean {

    private String[] arrs;
    private List<String> list;
    private Set<String> set;
    private Map<String, String> map;

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

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

    public void setSet(Set<String> set) {
        this.set = set;
    }

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

    @Override
    public String toString() {
        return "CollectionBean [arrs=" + Arrays.toString(arrs) + ", list=" +
list + ", set=" + set + ", map=" + map
                + "]";
    }

}
```

配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
```

```
    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="collectionBean" class="com.admiral.spring.demo4.CollectionBean">
        <!-- 注入数组 -->
        <property name="arrs">
            <list>
                <value>小白白</value>
                <value>小嘿嘿</value>
                <value>小黄黄</value>
            </list>
        </property>
        <!-- 注入List -->
        <property name="list">
            <list>
                <value>张三</value>
                <value>李四</value>
                <value>王五</value>
            </list>
        </property>
        <!-- 注入Set -->
        <property name="set">
            <set>
                <value>AAA</value>
                <value>BBB</value>
                <value>CCC</value>
            </set>
        </property>
        <!-- 注入Map -->
        <property name="map">
            <map>
                <entry key="aaa" value="111" />
                <entry key="bbb" value="222" />
                <entry key="ccc" value="333" />
            </map>
        </property>
    </bean>


</beans>
```

测试类:

```
/**
* @Title: SpringDemo4.java
* @Package com.admiral.spring.demo4
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
```

```java
package com.admiral.spring.demo4;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDemo4 {

    @Test
    public void demo1() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        CollectionBean collectionBean = (CollectionBean)
applicationContext.getBean("collectionBean");
        System.out.println(collectionBean);
    }
}
```

1.2.6.7 Spring的分配置文件的开发

1.2.6.7.1 在加载配置文件的时候，加载多个

```java
    @Test
    public void demo1() {
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml",
                "applicationContext2.xml");
        CollectionBean collectionBean = (CollectionBean) applicationContext.getBean("collectionBean");
        System.out.println(collectionBean);
    }
}
```

1.2.6.7.2 在一个配置文件中引入多个配置文件

```xml
<import resource="applicationContext2.xml"/>
```

1.3 案例代码

1.3.1 环境搭建

1.3.1.0 创建数据库和表

1.3.1.1 创建web项目，引入jar包

1.3.1.2 引入配置文件

1.3.1.4 创建包结构和类

1.3.1.5 在添加页面提交内容到Action:

1.3.1.6 改写 Action 类并配置 Action

```
/**
* @Title: CustomerAction.java
* @Package com.admiral.web.action
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.web.action;

import com.admiral.domain.Customer;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

public class CustomerAction extends ActionSupport implements
ModelDriven<Customer>{

    private Customer customer = new Customer();

    @Override
    public Customer getModel() {
        return customer;
    }

    /**
     * saveUI:跳转到添加页面的方法
     */
    public String saveUI() {

        return "saveUI";
    }

}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
```

```
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <package name="crm" extends="struts-default" namespace="/">
        <action name="customer_*" class="com.admiral.web.action.CustomerAction"
method="{1}">
            <result name="saveUI">/jsp/customer/add.jsp</result>
        </action>
    </package>

</struts>
```

1.3.1.7 在Action调用业务层:

1.3.2 Spring 整合 WEB 项目

1.3.2.1 引入 spring-web.jar 包

1.3.2.2 改写 Action

CustomerAction.java

```
/**
* @Title: CustomerAction.java
* @Package com.admiral.web.action
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.web.action;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.admiral.domain.Customer;
import com.admiral.service.CustomerService;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

public class CustomerAction extends ActionSupport implements
ModelDriven<Customer>{

    private Customer customer = new Customer();

    @Override
    public Customer getModel() {
```

```
        return customer;
    }

    /**
     * saveUI:跳转到添加页面的方法
     */
    public String saveUI() {

        return "saveUI";
    }
    /**
     * save:编写保存客户的方法
     */
    public String save() {
        //创建 Spring 工厂
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        CustomerService customerService = (CustomerService)
applicationContext.getBean("customerService");
        System.out.println("CustomerAction 中的 save 方法执行了......");
        customerService.save(customer);
        return NONE;
    }

}
```

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <package name="crm" extends="struts-default" namespace="/">
        <action name="customer_*" class="com.admiral.web.action.CustomerAction"
method="{1}">
            <result name="saveUI">/jsp/customer/add.jsp</result>
        </action>
    </package>

</struts>
```

1.3.2.3 编写Dao并配置

CustomerServiceImpl.java

```
/**
* @Title: CustomerServiceImpl.java
* @Package com.admiral.service.impl
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.service.impl;

import com.admiral.dao.CustomerDao;
import com.admiral.domain.Customer;
import com.admiral.service.CustomerService;

public class CustomerServiceImpl implements CustomerService {

    private CustomerDao customerDao;

    public void setCustomerDao(CustomerDao customerDao) {
        this.customerDao = customerDao;
    }

    @Override
    public void save(Customer customer) {
        System.out.println("CustomerServiceImpl 的 save 方法执行了......");
        customerDao.save(customer);
    }

}
```

CustomerDaoImpl.java

```
/**
* @Title: CustomerDaoImpl.java
* @Package com.admiral.dao.impl
* @Description:
* @author 白世鑫
* @date 2020-10-9
* @version V1.0
*/
package com.admiral.dao.impl;

import com.admiral.dao.CustomerDao;
import com.admiral.domain.Customer;

public class CustomerDaoImpl implements CustomerDao {
```

```
    @Override
    public void save(Customer customer) {
        System.out.println("CustomerDaoImpl 中的 save 方法执行了......");
    }

}
```

applicationContext.xml

```
<?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="customerService"
class="com.admiral.service.impl.CustomerServiceImpl">
        <property name="customerDao" ref="customerDao" />
    </bean>

    <bean id="customerDao" class="com.admiral.dao.impl.CustomerDaoImpl">
    </bean>

</beans>
```

1.3.2.4 业务层调用DAO