Spring_day01 总结

今日任务

- 使用 Spring 完成对客户的保存操作

教学导航

教学目标	
教学方法	案例驱动法

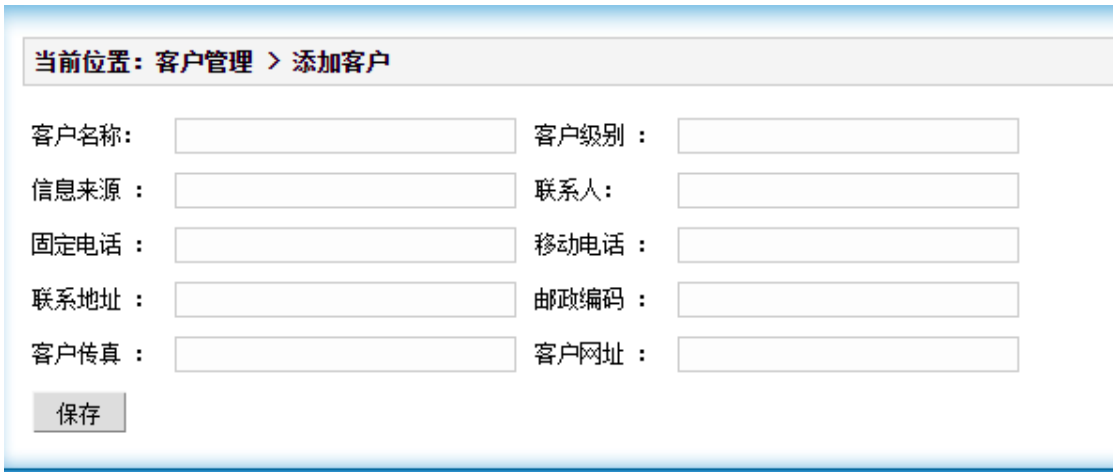
案例一使用Spring的IOC完成保存客户的操作:

1.1 案例需求

1.1.1 需求概述

CRM 系统中客户信息管理模块功能包括：
新增客户信息
客户信息查询
修改客户信息
删除客户信息

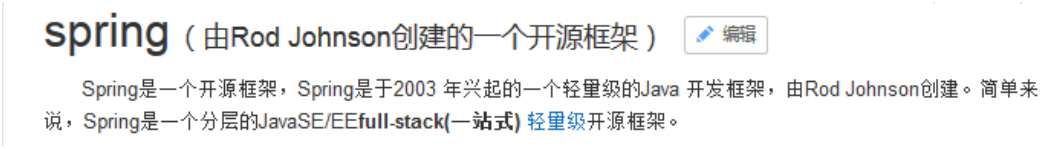
本功能要实现新增客户，页面如下：



1.2相关知识点

1.1.1 Spring 的概述:

1.2.1.1 什么是 Spring :



Spring 是一个开源框架，Spring 是于 2003 年兴起的一个轻量级的 Java 开发框架，由 Rod Johnson 在其著作 Expert One-On-One J2EE Development and Design 中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许使用者选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。Spring 使用基本的 JavaBean 来完成以前只可能由 EJB 完成的事情。然而，Spring 的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何 Java 应用都可以从 Spring 中受益。Spring 的核心是控制反转（IoC）和面向切面（AOP）。简单来说，**Spring 是一个分层的 JavaSE/EEfull-stack(一站式) 轻量级开源框架。**

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.ORM 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 JavaEE API 的使用难度

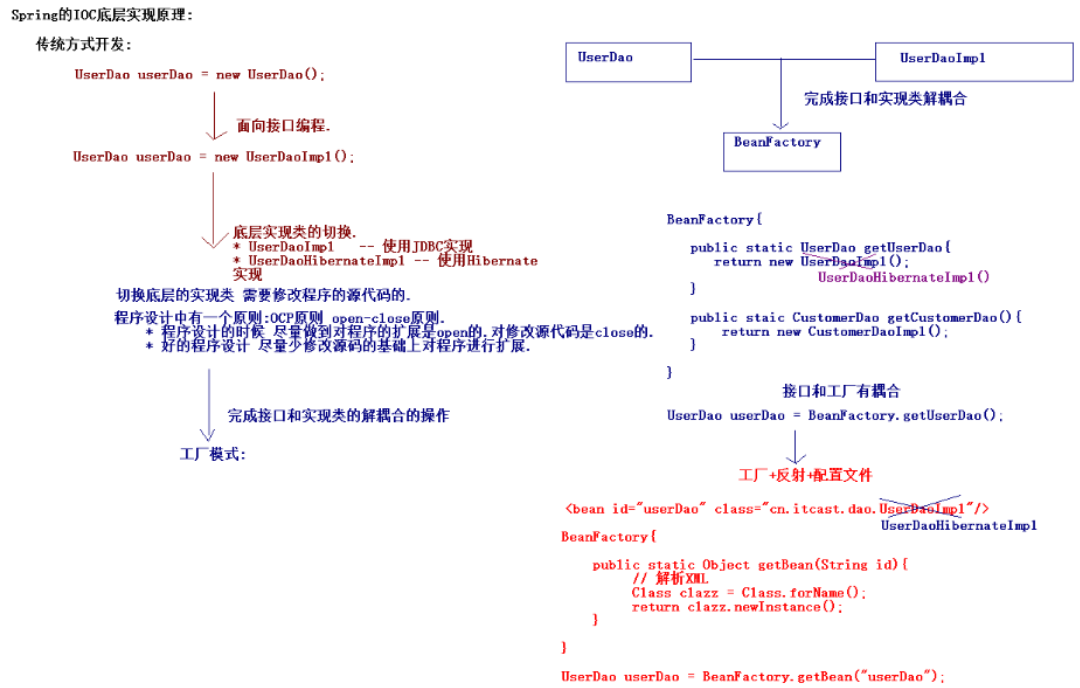
Spring 对 JavaEE 开发中非常难用的一些 API（JDBC、JavaMail、远程调用等），都提供了封装，使这些 API 应用难度大大降低

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



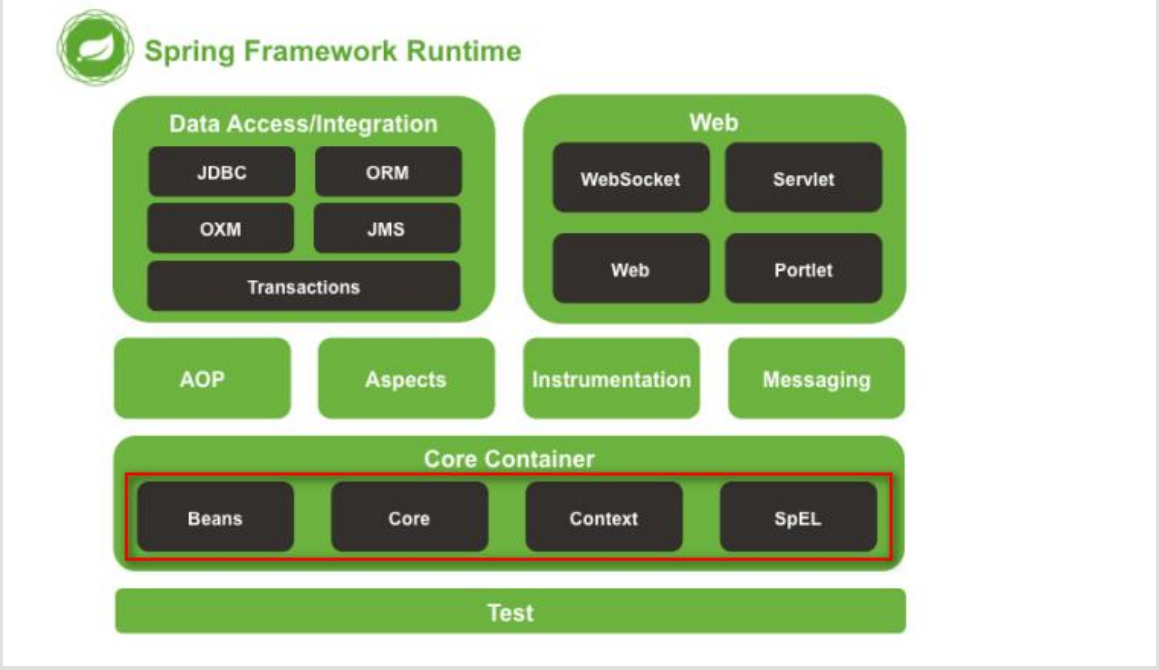
IOC：Inversion of Control 控制反转. 指的是 对象的创建权反转(交给)给 Spring.
作用是实现了程序的解耦合.

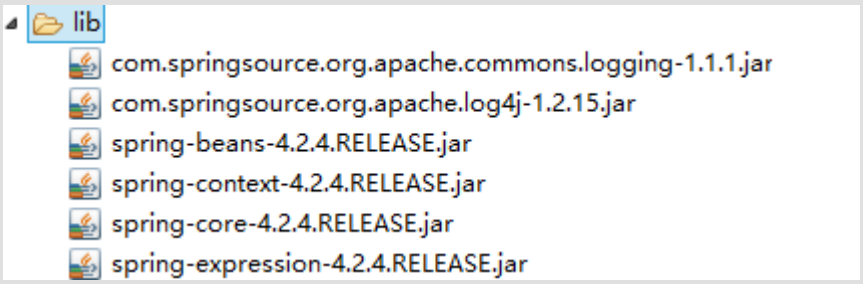
1.2.2.2 步骤一:下载 Spring 的开发包:

官网：http://spring.io/
下载地址：http://repo.springsource.org/libs-release-local/org/springframework/spring 解压:(Spring 目录结构:)

* docs :API 和开发规范.
* libs :jar 包和源码.
* schema :约束.

1.2.2.3 步骤二:创建 web 项目,引入 Spring 的开发包:





1.2.2.4 步骤三:引入相关配置文件:

```
    log4j.properties
    applicationContext.xml
    引入约束:
    spring-framework-4.2.4.RELEASE\docs\spring-framework-reference\html\xsd-configur
ation.html

    <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">

    </beans>
```

1.2.2.5 步骤四:编写相关的类:

```
public interface UserDao {

    public void sayHello();
}

public class UserDaoImpl implements UserDao {

    @Override
    public void sayHello() {
        System.out.println("Hello Spring...");
    }

}
```

1.2.2.6 步骤五:完成配置:

```
<!-- Spring 的入门案例================ -->
<bean id="userDao" class="cn.itcast.spring.demo1.UserDaoImpl"></bean>
```

1.2.2.7 步骤六:编写测试程序:

```
@Test
// Spring 的方式:
public void demo2(){
    // 创建 Spring 的工厂类:
    ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
    // 通过工厂解析 XML 获取 Bean 的实例.
    UserDao userDao = (UserDao) applicationContext.getBean("userDao");
    userDao.sayHello();
}
```

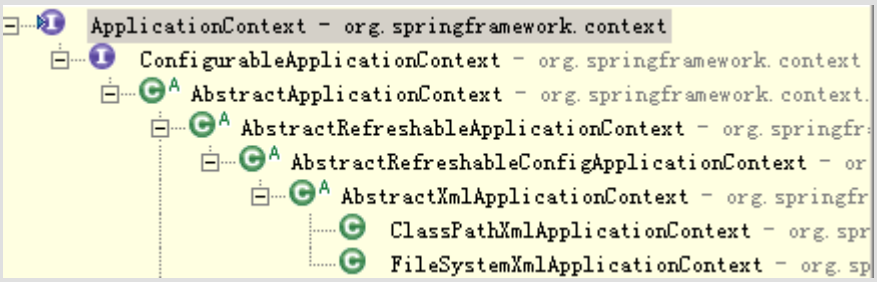
1.2.2.8 IOC 和 DI:

IOC :控制反转,将对象的创建权交给了 Spring.
DI :Dependency Injection 依赖注入.需要有 IOC 的环境,Spring 创建这个类的过程中,Spring 将类的依赖的属性设置进去.

1.2.3 Spring 中的工厂(容器):

1.2.3.1 ApplicationContext:

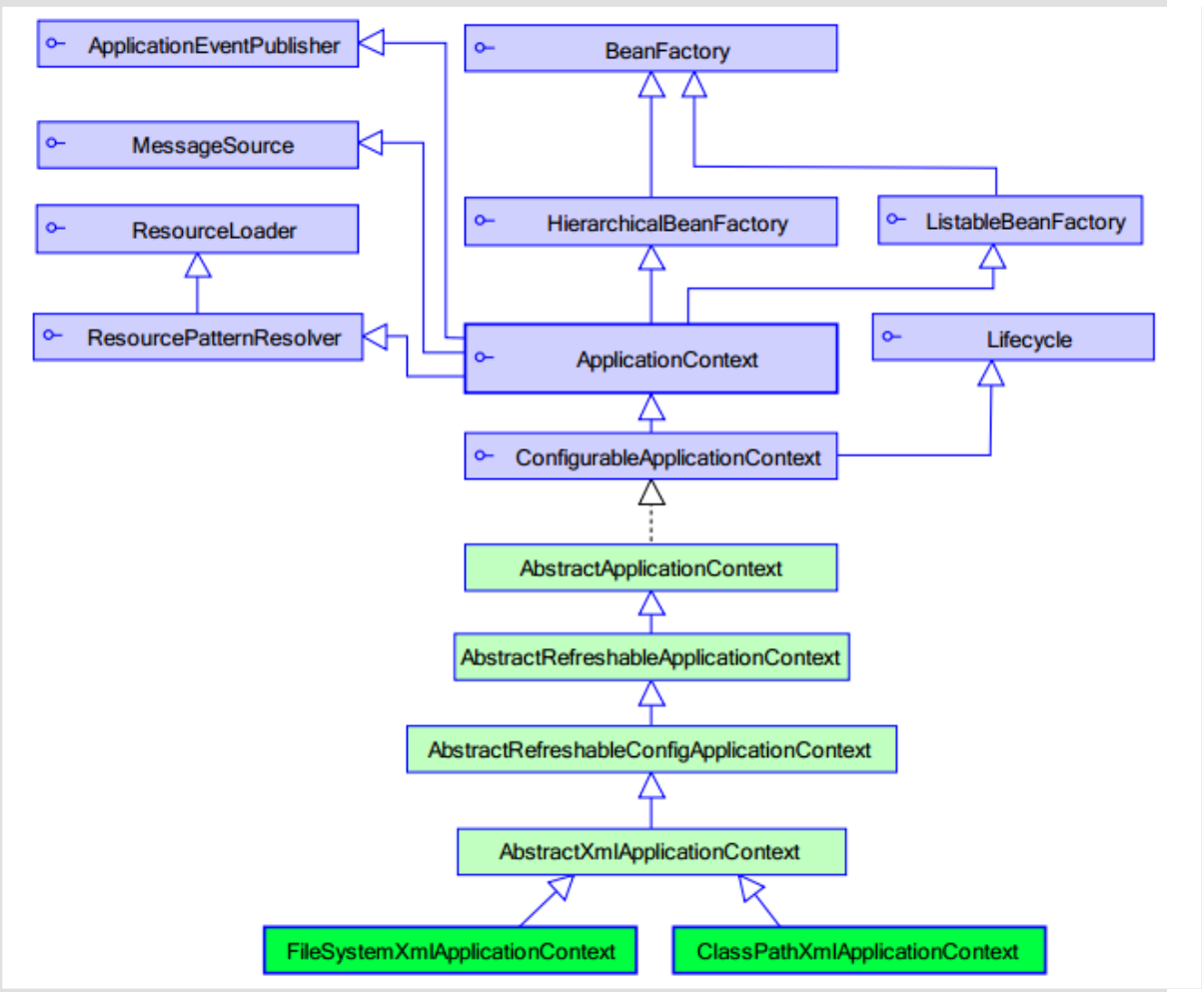
ApplicatioContext 接口有两个实现类:



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(过时):



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(容器启动)时候就会创建.
```

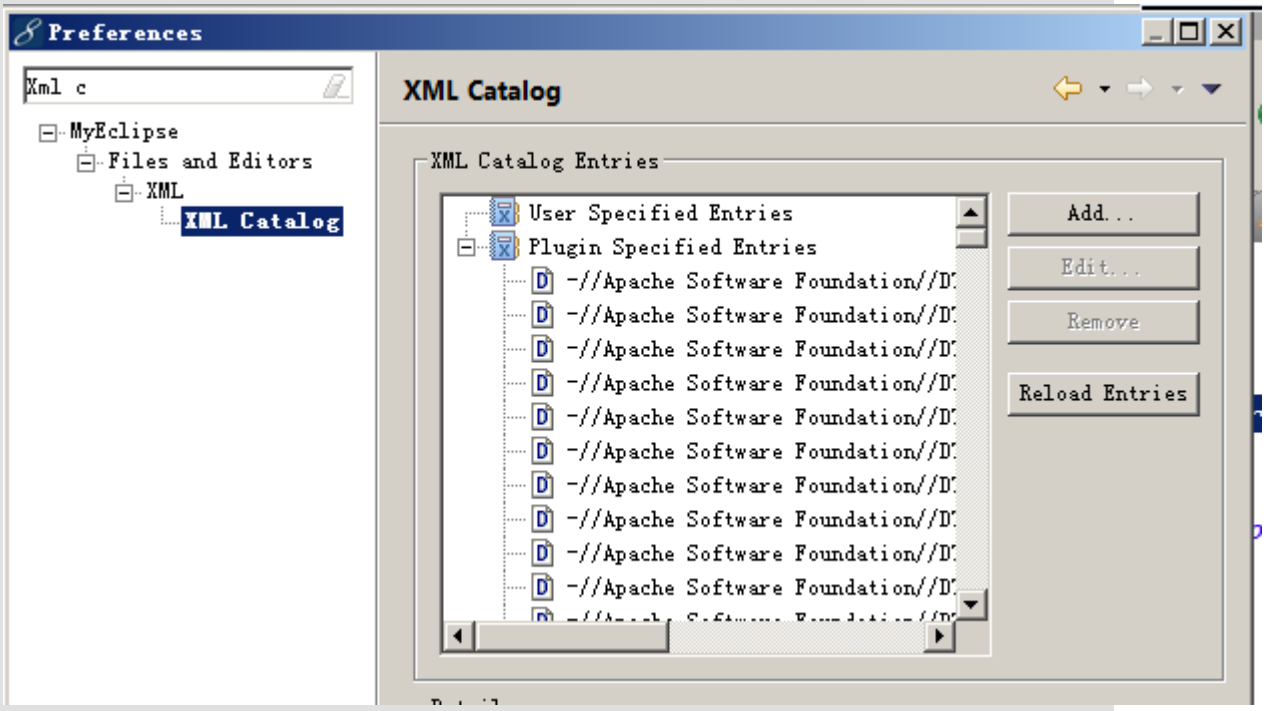
1.2.4 配置 STS 的 XML 的提示:

1.2.4.1 Spring 配置文件中提示的配置

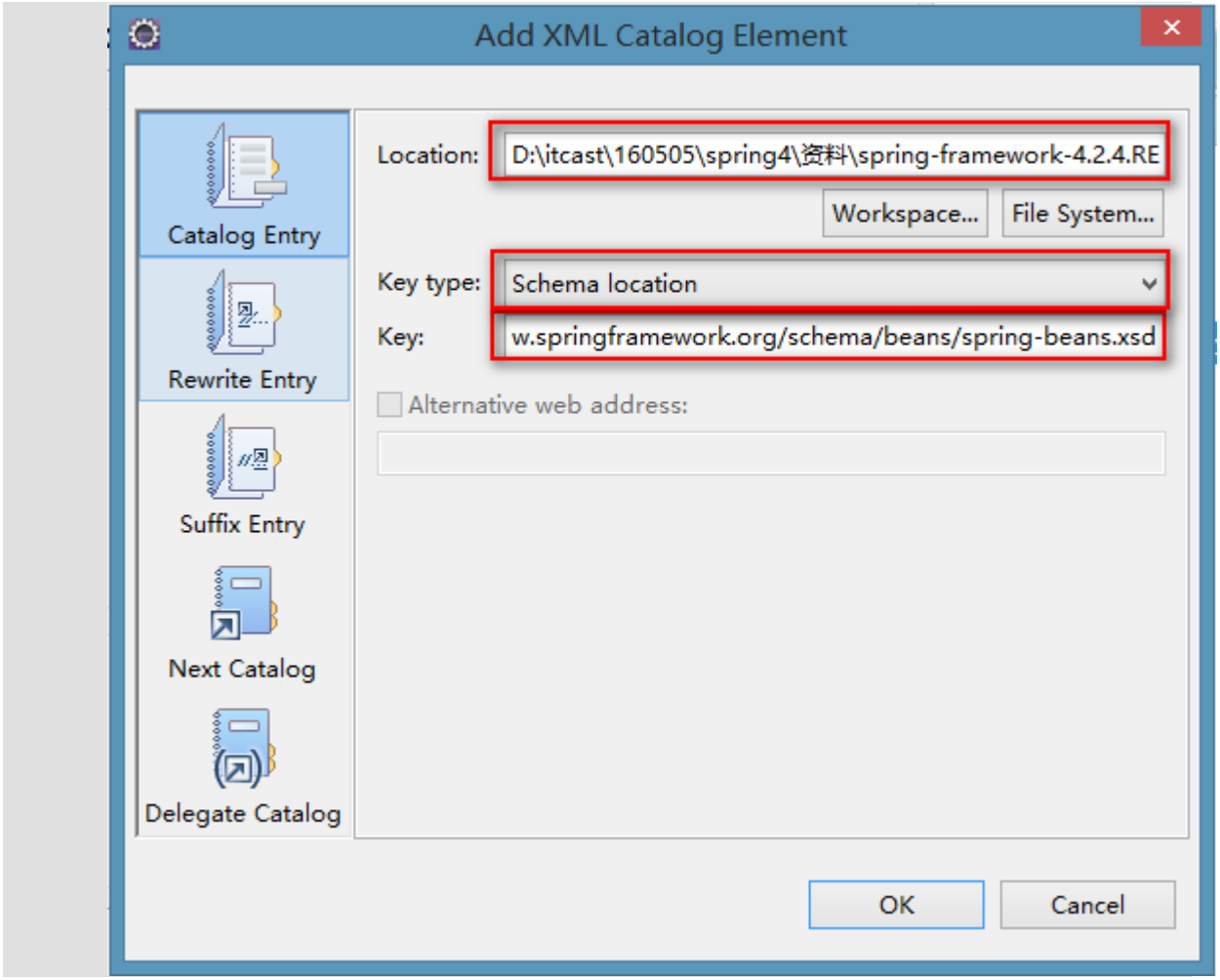
复制路径:

* http://www.springframework.org/schema/beans/spring-beans.xsd

查找 XML Catalog:



点击 Add..



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   :Bean 起个名字.   在约束中采用 ID 的约束:唯一.必须以字母开始，可以使用字母、数字、连字符、下划线、句话、冒号 id:不能出现特殊字符.
    <bean id=”bookAction”>
    name:Bean 起个名字.   没有采用 ID 的约束. name:出现特殊字符.如果<bean>没有 id 的话 ， name 可以当做 id 使用.
    * 整合 struts1 的时候:
    <bean name=”/loginAction” >
```

1.2.5.2 scope 属性：Bean 的作用范围.

```
    * singleton      :默认值，单例的.
    * prototype      :多例的.
```

```
    * request       :WEB 项目中,Spring 创建一个 Bean 的对象,将对象存入到 request 域中.
    * session       :WEB 项目中,Spring 创建一个 Bean 的对象,将对象存入到 session 域中.
    * globalSession :WEB 项目中,应用在 Porlet 环境.如果没有 Porlet 环境那么 globalSession 相当于 session.
```

1.2.5.3 Bean 的生命周期的配置:

```
    通过配置<bean>标签上的 init-method 作为 Bean 的初始化的时候执行的方法，配置 destroy-method 作为 Bean 的销毁的时候执行的方法。
    销毁方法想要执行，需要是单例创建的 Bean 而且在工厂关闭的时候，Bean 才会被销毁.
```

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)

【无参数的构造方法的方式:】

```
    <!-- 方式一：无参数的构造方法的实例化 -->
    <bean id="bean1" class="cn.itcast.spring.demo3.Bean1"></bean>
```

【静态工厂实例化的方式】

```
    提供一个工厂类:
    public class Bean2Factory {

        public static Bean2 getBean2(){
            return new Bean2();
        }

    }

    <!-- 方式二：静态工厂实例化 Bean -->
    <bean            id="bean2"           class="cn.itcast.spring.demo3.Bean2Factory" factory-method="getBean2"/>
```

【实例工厂实例化的方式】

```
    提供 Bean3 的实例工厂:
    public class Bean3Factory {

        public Bean3 getBean3(){
            return new Bean3();
        }
    }

    <!-- 方式三：实例工厂实例化 Bean -->
```

```
    <bean id="bean3Factory" class="cn.itcast.spring.demo3.Bean3Factory"></bean>
    <bean id="bean3" factory-bean="bean3Factory" factory-method="getBean3"></bean>
```

1.2.6.2 Spring 的 Bean 的属性注入:

【构造方法的方式注入属性】

```
        <!-- 第一种：构造方法的方式 -->
        <bean id="car" class="cn.itcast.spring.demo4.Car">
            <constructor-arg name="name" value="保时捷"/>
            <constructor-arg name="price" value="1000000"/>
        </bean>
```

【set 方法的方式注入属性】

```
        <!-- 第二种：set 方法的方式 -->
        <bean id="car2" class="cn.itcast.spring.demo4.Car2">
            <property name="name" value="奇瑞 QQ"/>
            <property name="price" value="40000"/>
        </bean>
```

1.2.6.3 Spring 的属性注入：对象类型的注入:

```
        <!-- 注入对象类型的属性 -->
        <bean id="person" class="cn.itcast.spring.demo4.Person">
            <property name="name" value="会希"/>
            <!-- ref 属性：引用另一个 bean 的 id 或 name -->
            <property name="car2" ref="car2"/>
        </bean>
```

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.

```
    第一步:引入 p 名称空间
    <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">

    第二步:使用 p 名称空间.
        * 普通属性:              p:属性名称=""
        * 对象类型属性:     p:属性名称-ref=""
```

```
    <!-- p 名称空间的属性注入的方式 -->
        <bean   id="car2"   class="cn.itcast.spring.demo4.Car2"   p:name="宝马 7"
p:price="1200000"/>
    <bean   id="person"   class="cn.itcast.spring.demo4.Person"   p:name="思聪 "
p:car2-ref="car2"/>
```

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

```
    SpEL: Spring Expression Language.
    语法:#{ SpEL }

        <!-- SpEL 的注入的方式 -->
        <bean id="car2" class="cn.itcast.spring.demo4.Car2">
            <property name="name" value="#{'奔驰'}"/>
            <property name="price" value="#{800000}"/>
        </bean>

       <bean id="person" class="cn.itcast.spring.demo4.Person">
        <property name="name" value="#{'冠希'}"/>
        <property name="car2" value="#{car2}"/>
       </bean>

        <bean id="carInfo" class="cn.itcast.spring.demo4.CarInfo"></bean>

        引用了另一个类的属性
        <bean id="car2" class="cn.itcast.spring.demo4.Car2">
    <!--        <property name="name" value="#{'奔驰'}"/> -->
            <property name="name" value="#{carInfo.carName}"/>
            <property name="price" value="#{carInfo.calculatePrice()}"/>
        </bean>
```

1.2.6.6 注入复杂类型:

```
    <!-- Spring 的复杂类型的注入==================== -->
        <bean id="collectionBean" class="cn.itcast.spring.demo5.CollectionBean">
            <!-- 数组类型的属性 -->
            <property name="arrs">
                <list>
                    <value>会希</value>
                    <value>冠希</value>
                    <value>天一</value>
                </list>
```

```
        </property>

        <!-- 注入 List 集合的数据 -->
        <property name="list">
            <list>
                <value>芙蓉</value>
                <value>如花</value>
                <value>凤姐</value>
            </list>
        </property>

        <!-- 注入 Map 集合 -->
        <property name="map">
            <map>
                <entry key="aaa" value="111"/>
                <entry key="bbb" value="222"/>
                <entry key="ccc" value="333"/>
            </map>
        </property>

        <!-- Properties 的注入 -->
        <property name="properties">
            <props>
                <prop key="username">root</prop>
                <prop key="password">123</prop>
            </props>
        </property>
    </bean>
```

1.2.6.7 Spring 的分配置文件的开发

```
    一种:创建工厂的时候加载多个配置文件:
    ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml","applicationContext2.xml");

    二种:在一个配置文件中包含另一个配置文件:
    <import resource="applicationContext2.xml"></import>
```

1.3 案例代码

1.3.1 搭建环境:

1.3.1.1 创建 web 项目，引入 jar 包.

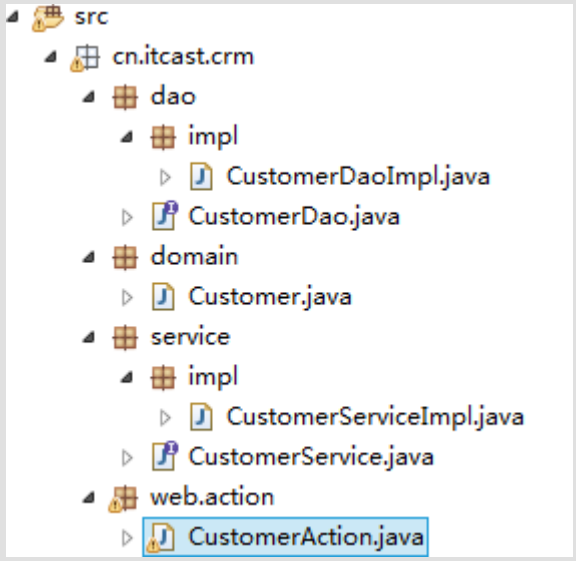
```
WEB 层使用 Struts2:
    * Struts2 开发的基本的包
Spring 进行 Bean 管理:
    * Spring 开发的基本的包
```

1.3.1.2 引入配置文件:

```
Struts2:
   * web.xml
  <filter>
    <filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFil
ter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
    * struts.xml

Spring:
   * applicationContext.xml
   * log4j.properties
```

1.3.1.3 引入页面:

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:

```
<FORM id=form1 name=form1
    action="${pageContext.request.contextPath }/customer_save.action"
    method=post>
```

1.3.1.6 改写 Action 类并配置 Action：

```
    <struts>
        <package name="crm" extends="struts-default" namespace="/">
            <action name="customer_*" class="cn.itcast.crm.web.action.CustomerAction"
method="{1}">

            </action>
        </package>
    </struts>
```

1.3.1.7 在 Action 调用业务层:

```
    将业务层类配置到 Spring 中:
        <bean                                                    id="customerService"
class="cn.itcast.crm.service.impl.CustomerServiceImpl">

        </bean>
```

```
在 Action 中获取业务层类:
    public String save(){
        System.out.println("Action 中的 save 方法执行了...");
        System.out.println(customer);

        // 传统方式:
        /*CustomerService customerService = new CustomerServiceImpl();
        customerService.save(customer);*/

        // Spring 的方式进行操作:
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
        CustomerService customerService = (CustomerService)
applicationContext.getBean("customerService");
        customerService.save(customer);
        return NONE;
    }

**** 每次请求都会创建一个工厂类,服务器端的资源就浪费了,一般情况下一个工程只有一个 Spring 的工厂
类就 OK 了.
    * 将工厂在服务器启动的时候创建好,将这个工厂放入到 ServletContext 域中.每次获取工厂从
ServletContext 域中进行获取.
    * ServletContextLinstener :监听 ServletContext 对象的创建和销毁.
```

1.3.2 Spring 整合 WEB 项目

1.3.2.1 引入 spring-web.jar 包:

```
配置监听器:
  <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
  </context-param>
```

1.3.2.2 改写 Action:

```
        /**
         * 保存客户的执行的方法：save
         */
        public String save(){
            // 传统方式：
            /*CustomerService customerService = new CustomerServiceImpl();
            customerService.save(customer);*/

            // Spring 的方式进行操作：
            /*ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
            CustomerService customerService = (CustomerService)
applicationContext.getBean("customerService");*/

            WebApplicationContext applicationContext
=WebApplicationContextUtils.getWebApplicationContext(ServletActionContext.getServle
tContext());
            CustomerService customerService = (CustomerService)
applicationContext.getBean("customerService");

            System.out.println("Action 中的 save 方法执行了...");
            System.out.println(customer);
            customerService.save(customer);
            return NONE;
        }
```

1.3.2.3 编写 Dao 并配置:

```
        <bean id="customerDao" class="cn.itcast.crm.dao.impl.CustomerDaoImpl">

        </bean>
```

1.3.2.4 业务层调用 DAO：

```
    public class CustomerServiceImpl implements CustomerService {

        private CustomerDao customerDao;
```

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

    …
    }

        <bean                                                    id="customerService"
class="cn.itcast.crm.service.impl.CustomerServiceImpl">
            <property name="customerDao" ref="customerDao"/>
        </bean>
```