spring学习日志一

一、spring介绍

1.1 spring简介

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.

1.2 为啥学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

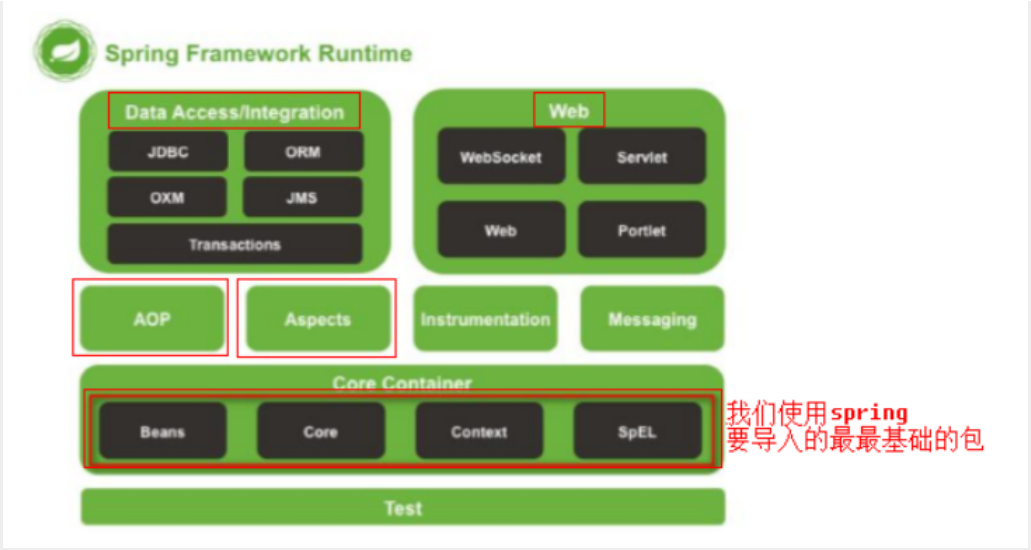
Spring 不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz 等）的直接支持

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

二、spring环境搭建

2.1 导包



2.2 创建一个对象

```java
package cn.itcast.bean;

public class User {
    private String name;
    private Integer age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }

}
```

2.3 书写配置文件注册对象到容器

位置任意(建议放到src下)
配置文件名任意(建议applicationContext.xml)

```
<!-- 将User对象交给spring容器管理 -->
<bean name="user" class="cn.itcast.bean.User" ></bean>
```

2.4 测试

```
@Test
public void fun1(){

    //1 创建容器对象
    ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
    //2 向容器"要"user对象
    User u = (User) ac.getBean("user");
    //3 打印user对象
    System.out.println(u);


}
```

三、spring的几个思想及概念

3.1 IOC思想

首先想说说 IoC（Inversion of Control 控制倒转）。这是 spring 的核心，贯穿始终。所谓 IoC，对于 spring 框架来说，就是由 spring 来负责控制对象的生命周期和对象间的关系。这是什么意思呢，举个简单的例子，我们是如何找女朋友的？常见的情况是，我们到处去看哪里有长得漂亮身材又好的mm，然后打听她们的兴趣爱好、qq号、电话号、ip号、iq号.....，想办法认识她们，投其所好送其所要，然后嘿嘿.....这个过程是复杂深奥的，我们必须自己设计和面对每个环节。传统的程序开发也是如此，在一个对象中，如果要使用另外的对象，就必须得到它（自己 new 一个，或者从 JNDI 中查询一个），使用完之后还要将对象销毁（比如 Connection 等），对象始终会和其他的接口或类藕合起来。

那么 IoC 是如何做的呢？有点像通过婚介找女朋友，在我和女朋友之间引入了一个第三者：婚姻介绍所。婚介管理了很多男男女女的资料，我可以向婚介提出一个列表，告诉它我想找个什么样的女朋友，比如长得像李嘉欣，身材像林熙雷，唱歌像周杰伦，速度像卡洛斯，技术像齐达内之类的，然后婚介就会按照我们的要求，提供一个mm，我们只需要去和她谈恋爱、结婚就行了。简单明了，如果婚介给我们的人选不符合要求，我们就会抛出异常。整个过程不再由我自己控制，而是有婚介这样一个类似容器的机构来控制。 Spring 所倡导的开发方式就是如此，所有的类都会在 spring 容器中登记，告诉 spring 你是个什么东西，你需要什么东西，然后 spring 会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring 来控制，也就是说控制对象生存周期的不再是引用它的对象，而是 spring 。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被 spring 控制，所以这叫控制反转。

IoC 的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过 DI （Dependency Injection，依赖注入）来实现的。比如 对象A 需要操作数据库，以前我们总是要在 A 中自己编写代码来获得一个 Connection 对象，有了 spring 我们就只需要告诉 spring， A 中需要一个 Connection，至于这个 Connection 怎么构造，何时构造， A 不需要知道。在系统运行时， spring 会在适当的时候制造一个 Connection，然后像打针一样，注射到 A 当中，这样就完成了对各个对象之间关系的控制。 A 需要依赖 Connection 才能正常运行，而这个 Connection 是由 spring 注入到 A 中的，依赖注入的名字就这么来的。

简而言之，就是把我们创建对象的方式反转了。以前对象的创建时由我们开发人员维护的。包括依赖关系也是自己注入的。使用spring之后，对象的创建以及依赖关系可以由spring完成创建以及注入。反转控制就是反转了对象的创建方式。从我们自己创建反转给了程序(spring)。

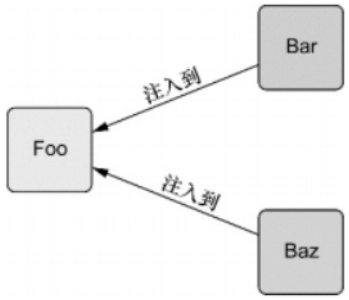
3.2 依赖性注入(DI,Dependency Injection)

依赖注入（DI）是控制反转（Ioc）的一种方式。依赖注入这个词让人望而生畏，现在已经演变成一项复杂的编程技巧 或设计模式理念。但事实证明，依赖注入并不像它听上去那么复杂。 在项目

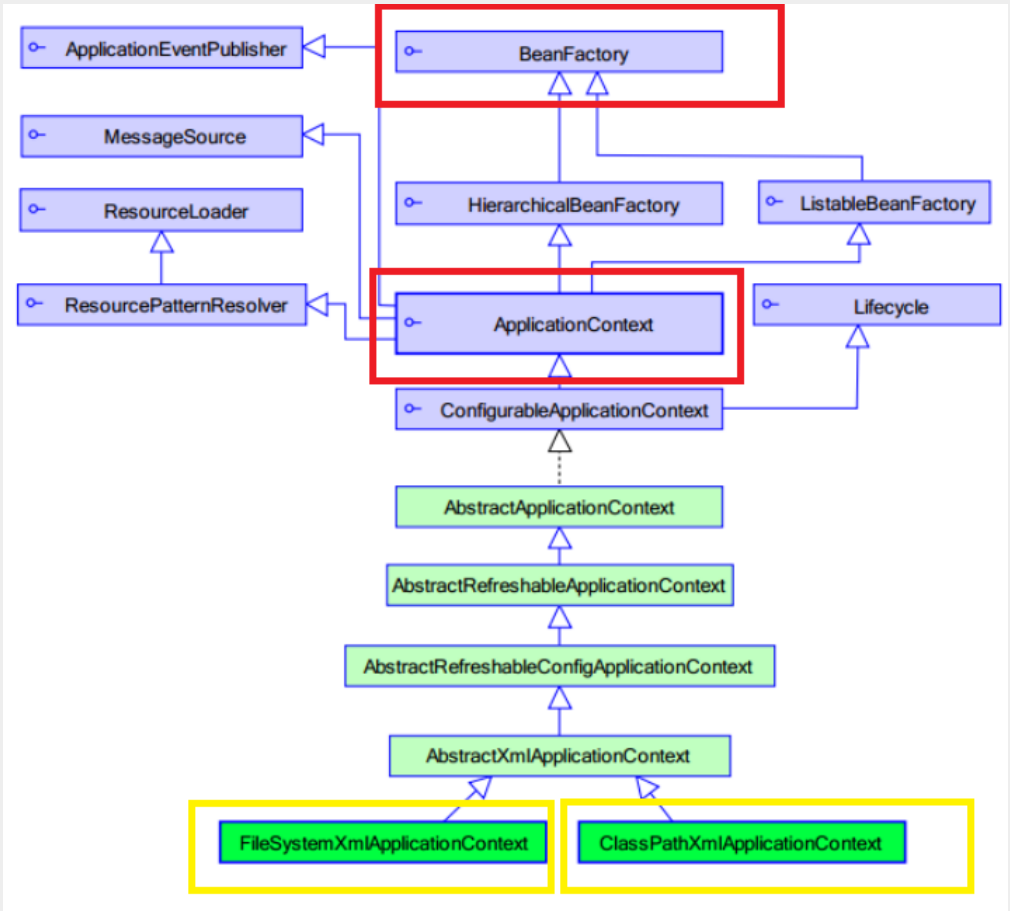
中应用DI，你会发现你的代码会变得异常简单并且更容易理解 和测试。DI功能是如何实现的任何一个有实际意义的应用（肯定比Hello World示例更复杂）都会由两个或者更多的类组成，这些类相互之间进行协作来完成特定的业务逻辑。按照传统的做法，每个对象负责管理与自己相互协作的对象（即它所依赖的对象）的引用，这将会导致高度耦合和难以测试的代码。

耦合具有两面性（two-headed beast）。一方面，紧密耦合的代码难以 测试、难以复用、难以理解 并且典型地表现出"打地鼠"式的bug特 性（修复一个bug，将会出现一个或者更多新的bug）。另一方面，一 定程度的耦合又是必须的——完全没有耦合的代码什么也做不了。为 了完成有实际意义的功能，不同的类必须以适当的方式进行交互。总 而言之，耦合是必须的，但应当被小心谨慎地管理。

通过DI，对象的依赖关系将由系统中负责协调各对象的第三方组件在 创建对象的时候进行设定。对象无需自行创建或管理它们的依赖关系，如图1.1所示，依赖关系将被自动注入到需要它们的对象当中 去。



3.3 BeanFactory接口与ApplicationContext接口与Bean元素



Spring有两个核心接口：`BeanFactory`和`ApplicationContext`，其中ApplicationContext是BeanFactory的子接口。

他们都可代表**Spring容器**。

Spring容器是生成Bean实例的工厂，并且管理容器中的Bean。
Bean是Spring管理的基本单位，在基于Spring的Java EE应用中，所有的组件都被当成Bean处理，包括数据源、Hibernate的SessionFactory、事务管理器等。在Spring中，Bean的是一个非常广义的概念，任何的Java对象、Java组件都被当成Bean处理。
而且应用中的所有组件，都处于Spring的管理下，都被Spring以Bean的方式管理，Spring负责创建Bean实例，并管理他们的生命周期。Bean在Spring容器中运行，无须感受Spring容器的存在，一样可以接受Spring的依赖注入，包括Bean属性的注入，协作者的注入、依赖关系的注入等。
Spring容器负责创建Bean实例，所以需要知道每个Bean的实现类，Java程序面向接口编程，无须关心Bean实例的实现类；但是Spring容器必须能够精确知道每个Bean实例的实现类，因此Spring配置文件必须精确配置Bean实例的实现类。

四、spring配置详解

业界一般命名为applicationContext.xml，位置任意。

4.1 bean的实例化方式

4.1.1 默认构造

只要创建了我们的实体类(bean，entity，这里其实可以弱化为任何类)，必有默认构造方法。
然后只需要我们在配置文件配置bean元素基本配置即可。

```
    <bean name="给类取的名字"  class="类的全路径名"></bean>
```

4.1.2 静态工厂

常用于spring整合其他框架(工具)

```java
import cn.scct.bean.User;

public class UserFactory {    直接产生一个对象而已

    public static User createUser(){

        System.out.println("静态工厂创建User");

        return new User();

    }
```

配置时要指定静态工厂类创建对象的静态方法：factory-method

```xml
<!-- 创建方式2:静态工厂创建
     调用UserFactory的createUser方法创建名为user2的对象，放入容器
 -->
<bean  name="user2"
    class="cn.scct.b_create.UserFactory"
    factory-method="createUser" ></bean>
```

测试代码都是一样的，下面不会再提

```java
public void fun1(){
    //1 创建容器对象
    ApplicationContext ac = new ClassPathXmlApplicationContext("cn/scct/b_create/applicationContext.xml");
    //2 向容器"要"user对象
    //User u = (User) ac.getBean("user");
    //不强制转换用下面这个方法
    User u =  ac.getBean("user",User.class);
                                        获取对象
    //3 打印user对象
    System.out.println(u);
}
```

src下的全路径

4.1.3 实例工厂

先有工厂实例对象，再有目标对象，不是静态方法

```java
import cn.scct.bean.User;
                                    实例工厂
public class UserFactory {    直接产生一个对象而已

    public User createUser2(){

        System.out.println("静态工厂创建User");

        return new User();

    }
```

```xml
<!-- 创建方式3:实例工厂创建                代表首先要创造工厂对象，
     调用UserFactory对象的createUser2方法创建名为user3的对象，放入容器
 -->                                再调用相应的方法得到目标对象即可
<bean  name="userFactory"
    class="cn.scct.b_create.UserFactory"    ></bean>
<bean  name="user3"
    factory-bean="userFactory"
    factory-method="createUser2" ></bean>
```

4.2 bean的种类

普通bean: 之前操作的都是普通bean。`<bean name="" class="A">`，spring直接创建A实例，并返回。
FactoryBean：是一个特殊的bean，具有工厂生成对象能力，只能生成特定的对象。这个bean实现 FactoryBean接口，并覆盖 getObject()，由getObject()的返回值特定bean。

```
    <bean name="" class="FB"> 先创建FB实例，使用调用getObject()方法，并返回方法的返回值
        FB fb = new FB();
        return fb.getObject();
```

BeanFactory 和 FactoryBean 对比

BeanFactory：**工厂**，用于**生成任意bean。**
FactoryBean：**特殊bean**，用于生成另一个特定的bean。例如：ProxyFactoryBean，此工厂bean用于生产代理。比如获得代理对象实例。AOP使用

4.3 bean的scope属性(作用域)

- 作用域：用于确定 spring 创建 bean 实例个数

类别	说明
singleton	在Spring IoC容器中仅存在一个Bean实例，Bean以单例方式存在，默认值
prototype	每次从容器中调用Bean时，都返回一个新的实例，即每次调用getBean()时，相当于执行new XxxBean()
request	每次HTTP请求都会创建一个新的Bean，该作用域仅适用于WebApplicationContext环境
session	同一个HTTP Session 共享一个Bean，不同Session使用不同Bean，仅适用于WebApplicationContext 环境
globalSession	一般用于Portlet应用环境，该作用域仅适用于WebApplicationContext 环境

- 取值：
  singleton 单例，默认值。
  prototype 多例，每执行一次 getBean 将获得一个实例。例如：struts 整合 spring，配置 action 多例。
- 配置信息

```
<bean id="" class=""   scope="">
```

```
    <bean id="userServiceId" clas
        scope="prototype" ></bean
```

prototype
request
session
singleton

```
<bean id="userServiceId" class="com.itheima.d_scope.UserServiceImpl"
      scope="prototype" ></bean>
```

4.4 bean的生命周期

一、在类中添加作为初始化和销毁的方法

```
public void myInit(){
    System.out.println("初始化");
}
public void myDestroy(){
    System.out.println("销毁");
}
```

二、配置

```
<!--
      init-method 用于配置初始化方法，准备数据等
      destroy-method 用于配置销毁方法，清理资源等
   -->
   <bean id="userServiceId" class="com.itheima.e_lifecycle.UserServiceImpl"
      init-method="myInit" destroy-method="myDestroy" ></bean>
```

三

```
public void demo02() throws Exception{
    //spring 工厂
    String xmlPath = "com/itheima/e_lifecycle/beans.xml";
    ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
    UserService userService = (UserService) applicationContext.getBean("userServiceId");
    userService.addUser();

    //要求：1.容器必须close，销毁方法执行；2.必须是单例的
    applicationContext.getClass().getMethod("close").invoke(applicationContext);
    // * 此方法接口中没有定义，实现类提供
    applicationContext.close();

}
```

测试时注意要关闭工厂

bean必须是单例的

4.5 spring的分模块配置

五、spring属性注入

5.1 set方式注入

```xml
<!-- set方式注入： --所谓属性注入就是为对象的属性注入值 这里算set方式注入
<bean  name="user" class="cn.scct.bean.User" >
    <!--值类型注入： 为User对象中名为name的属性注入tom作为值 --> 值类型注入
    <property name="name" value="tom" ></property>
    <property name="age"  value="18" ></property>
    <!-- 引用类型注入： 为car属性注入下方配置的car对象 -->
    <property name="car"  ref="car" ></property>
</bean>
```

引用类型注入

ref 为另一个bean的name值

当然，前提是注入类需要有set方法，不然你觉得为啥叫set注入

```xml
<!-- 将car对象配置到容器中 -->
<bean name="car" class="cn.scct.bean.Car" >
    <property name="name" value="兰博基尼" ></property>
    <property name="color" value="黄色" ></property>
</bean>
```

5.2 构造函数注入

```xml
<!-- ========================================================== -->
<!-- 构造函数注入 -->
<bean name="user2" class="cn.scct.bean.User" >
    <!-- name属性： 构造函数的参数名
    index属性： 构造函数的参数索引
    type属性： 构造函数的参数类型 -->
    <constructor-arg name="name" index="0" type="java.lang.Integer" value="999"  ></constructor-arg>
    <constructor-arg name="car" ref="car" index="1" ></constructor-arg>
</bean>
```

index 表示构造函数的参数值索引，值为0表示第一个参数，当然，

还要指定类型 和值(如果不是注入的bean的话用value属性)

如果是被注入的类型，用ref属性，当然也要标明index属性

显然，使用的构造函数是第二个

```java
public User(Car car,String name) {
    System.out.println("User(Car car,String name)!!");
    this.name = name;
    this.car = car;
}

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

5.3 p名称空间注入（了解）

```xml
-- p名称空间注入， 走set方法
  1.导入P名称空间 xmlns:p="http://www.springframework.org/schema/p"
  2.使用p:属性完成注入
      |-值类型： p:属性名="值"
      |-对象类型： p:属性名-ref="bean名称"
->
  <bean  name="user3" class="cn.scct.bean.User" p:name="jack" p:age="20" p:car-ref="car"  >
  </bean>
```

5.4 spel注入 （了解）

```
<!--
spel注入: spring Expression Language sping表达式语言
-->
<bean  name="user4" class="cn.scct.bean.User" >
        <property name="name" value="#{user.name}" ></property>
        <property name="age" value="#{user3.age}" ></property>
        <property name="car" ref="car" ></property>
</bean>
<!-- ============================================================ -->
```

5.5 复杂数据类型注入

先提供一个bean

```
public class CollectionBean {
    private Object[] arr;//数组类型注入
    private List list;//list/set 类型注入
    private Map map;//map类型注入
    private Properties prop;//properties类型注入
}
```

set和get方法略

5.5.1 数组类型注入

```
- 复杂类型注入 -->
<bean name="cb" class="cn.scct.c_injection.CollectionBean" >
    <!-- 如果数组中只准备注入一个值(对象),直接使用value|ref即可
    <property name="arr" value="tom" ></property>
    -->
    <!-- array注入,多个元素注入 -->
    <property name="arr">
        <array>
            <value>tom</value>
            <value>jerry</value>
            <ref bean="user4" />
        </array>
    </property>
```

//不是一个值或对象那么就得添加array属性

5.5.2 List类型注入

```
<!-- 如果List中只准备注入一个值(对象),直接使用value|ref即可
<property name="list" value="jack" ></property>-->
<property name="list"  >
    <list>
        <value>jack</value>
        <value>rose</value>
        <ref bean="user3" />
    </list>
</property>
```

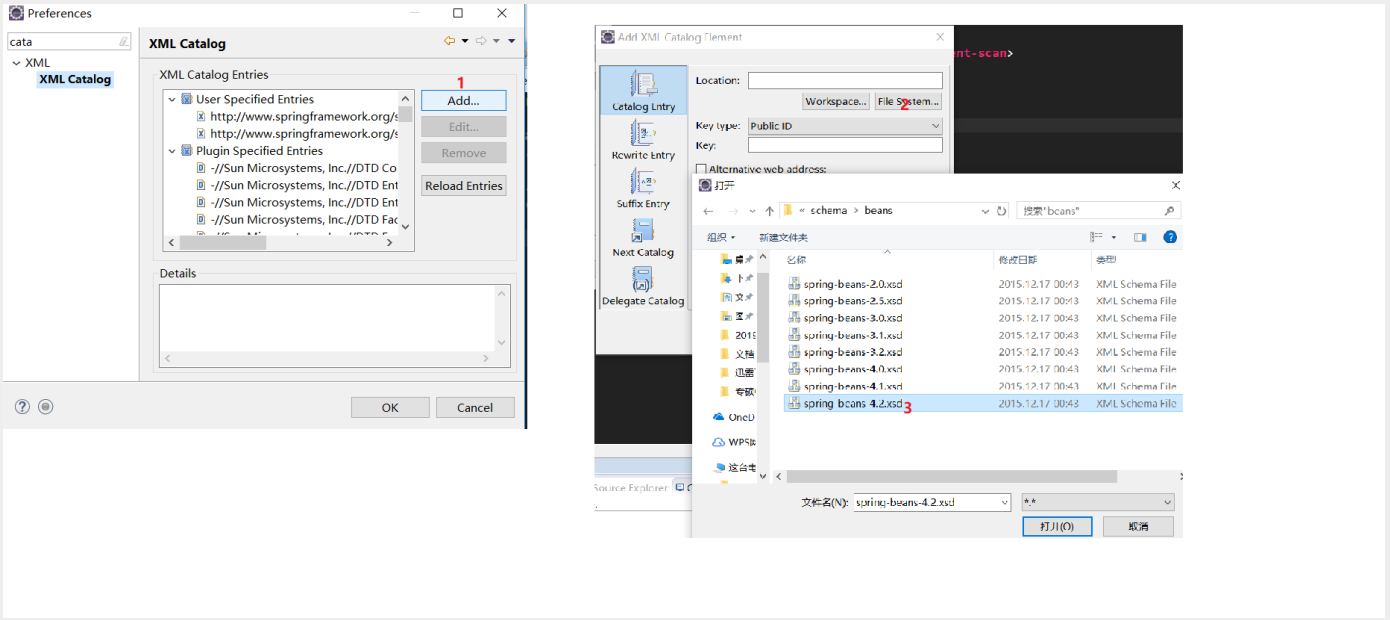
5.5.3 Map类型注入

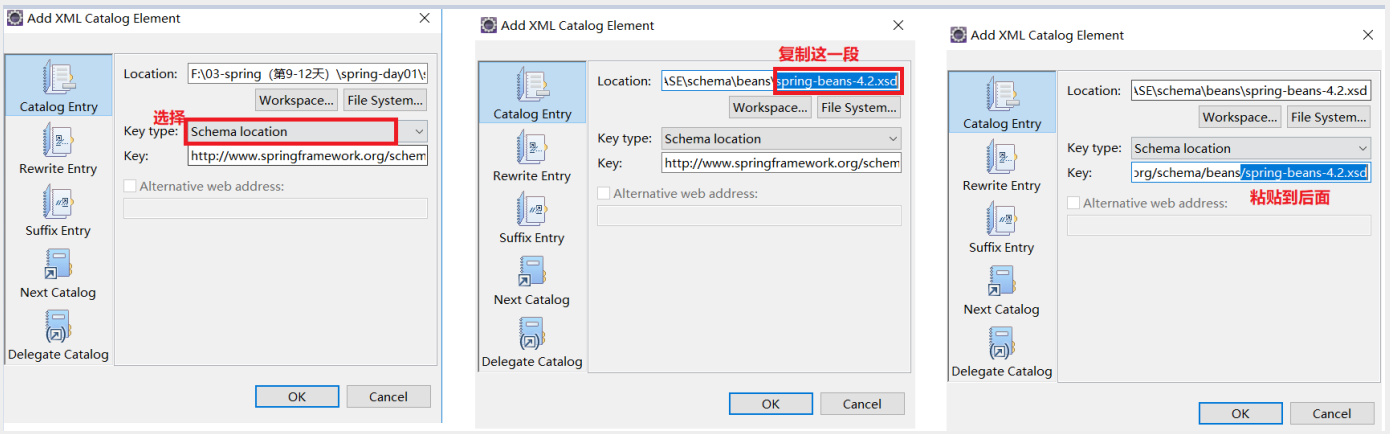
```
<!-- map类型注入 -->
<property name="map"  >
    <map>
        <entry key="url" value="jdbc:mysql:///crm" ></entry>
        <entry key="user" value-ref="user4"  ></entry>
        <entry key-ref="user3" value-ref="user2"  ></entry>
    </map>
</property>
```

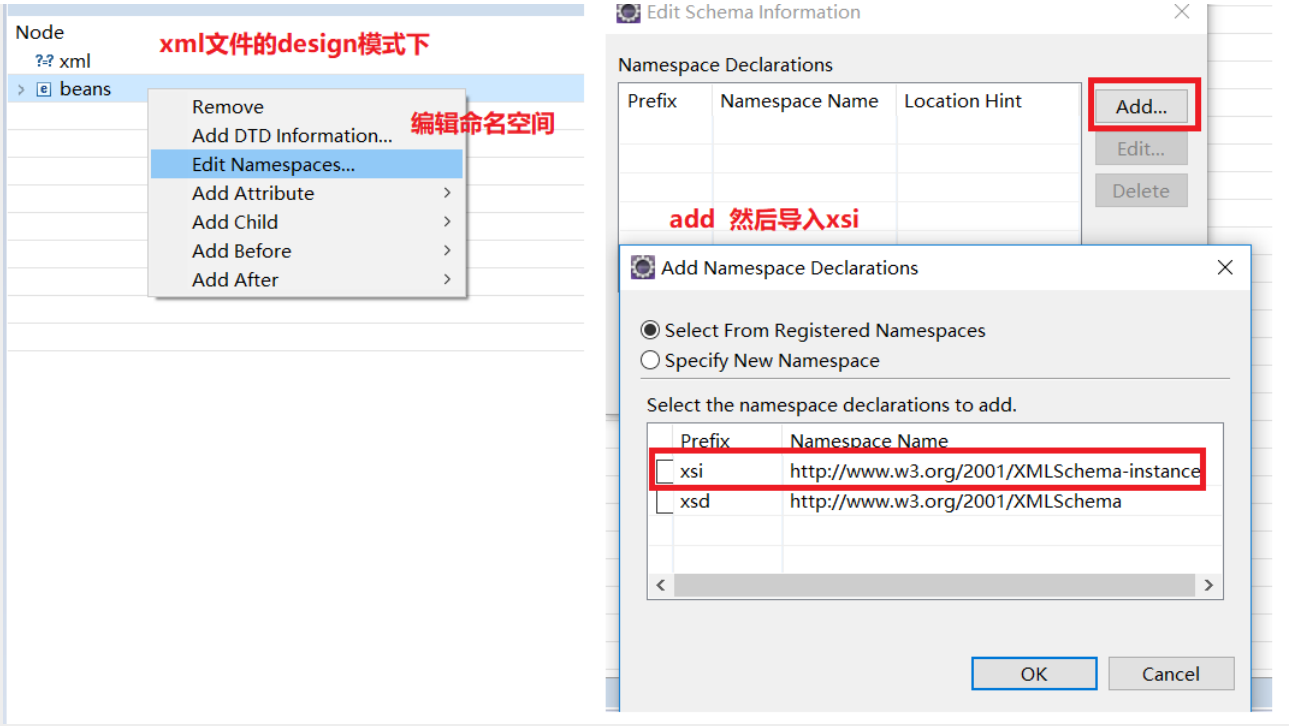
5.5.4 properties类型注入

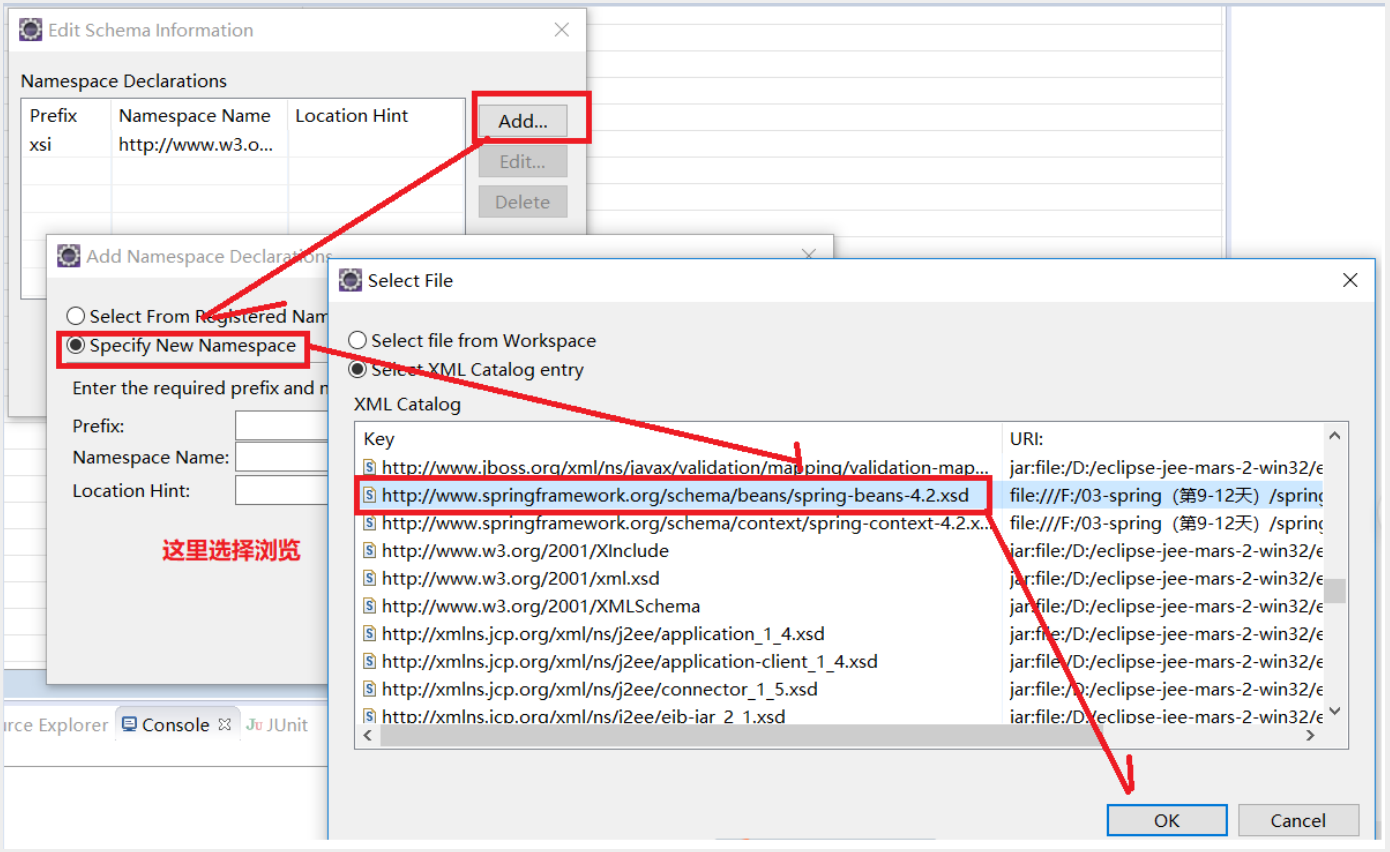
```
    <!-- properties 类型注入 -->
    <property name="prop"  >
        <props>
            <prop key="driverClass">com.jdbc.mysql.Driver</prop>
            <prop key="userName">root</prop>
            <prop key="password">1234</prop>
        </props>
    </property>
```

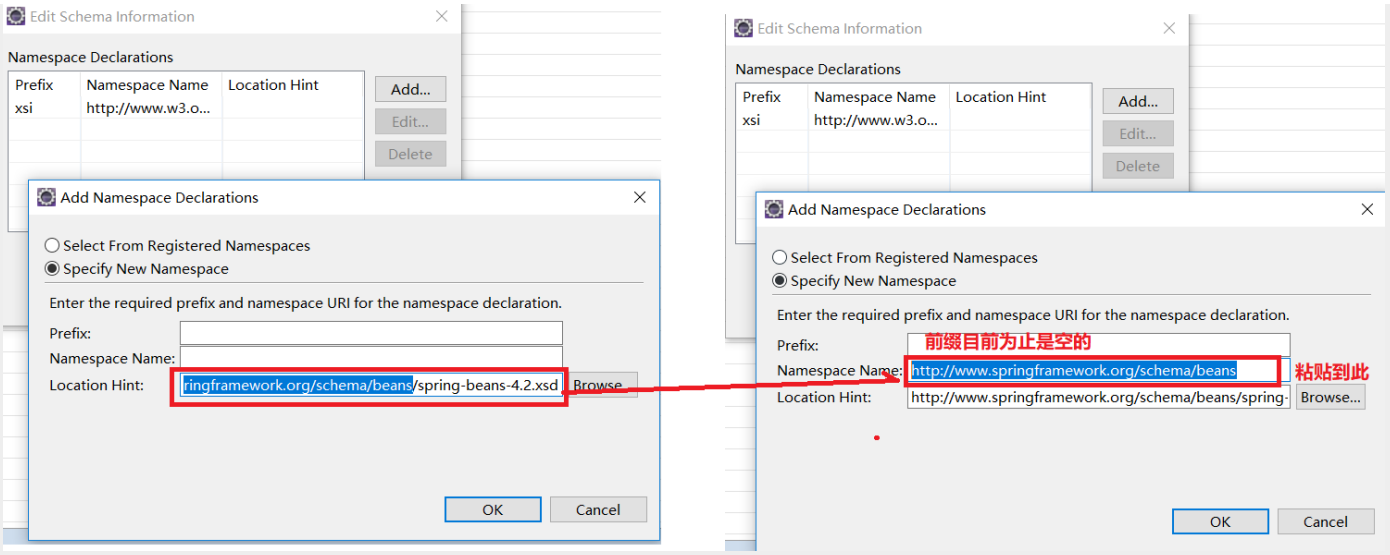
六、 导入约束步骤介绍






Node
xml
beans
xml文件的design模式下
Remove
Add DTD Information...
Edit Namespaces...
Add Attribute
Add Child
Add Before
Add After
编辑命名空间
Edit Schema Information
Namespace Declarations
Prefix
Namespace Name
Location Hint
Add...
Edit...
Delete
add 然后导入xsi
Add Namespace Declarations
Select From Registered Namespaces
Specify New Namespace
Select the namespace declarations to add.
Prefix
Namespace Name
xsi
http://www.w3.org/2001/XMLSchema-instance
xsd
http://www.w3.org/2001/XMLSchema
OK
Cancel


Edit Schema Information
Namespace Declarations
Prefix
Namespace Name
Location Hint
xsi
http://www.w3.o...
Add...
Edit...
Delete
Add Namespace Declarations
Select From Registered Nam
Specify New Namespace
Enter the required prefix and n
Prefix:
Namespace Name:
Location Hint:
这里选择浏览
Select File
Select file from Workspace
Select XML Catalog entry
XML Catalog
Key
URI:
http://www.jboss.org/xml/ns/javax/validation/mapping/validation-map...
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
file:///F:/03-spring（第9-12天）/spring
http://www.springframework.org/schema/context/spring-context-4.2.x...
file:///F:/03-spring（第9-12天）/spring
http://www.w3.org/2001/XInclude
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://www.w3.org/2001/xml.xsd
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://www.w3.org/2001/XMLSchema
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://xmlns.jcp.org/xml/ns/j2ee/application_1_4.xsd
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://xmlns.jcp.org/xml/ns/j2ee/application-client_1_4.xsd
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://xmlns.jcp.org/xml/ns/j2ee/connector_1_5.xsd
jar:file:/D:/eclipse-jee-mars-2-win32/e
http://xmlns.jcp.org/xml/ns/j2ee/ejb-jar_2_1.xsd
jar:file:/D:/eclipse-jee-mars-2-win32/e
OK
Cancel
Console
JUnit



七、谈依赖注入和控制反转的区别

DI—Dependency Injection，即“依赖注入”：组件之间依赖关系由容器在运行期决定，形象的说，即**由容器动态的将某个依赖关系注入到组件之中。依赖注入的目的并非为软件系统带来更多功能，而是为了提升组件重用的频率，并为系统搭建一个灵活、可扩展的平台。**通过依赖注入机制，我们只需要通过简单的配置，而无需任何代码就可指定目标需要的资源，完成自身的业务逻辑，而不需要关心具体的资源来自何处，由谁实现。

理解DI的关键是：“谁依赖谁，为什么需要依赖，谁注入谁，注入了什么”，那我们来深入分析一下：

- 谁依赖于谁：当然是**应用程序依赖于IoC容器**；
- 为什么需要依赖：**应用程序需要IoC容器来提供对象需要的外部资源**；
- 谁注入谁：很明显是**IoC容器注入应用程序某个对象，应用程序依赖的对象**；
- 注入了什么：就是**注入某个对象所需要的外部资源（包括对象、资源、常量数据）**。

IoC和DI由什么**关系**呢？其实它们**是同一个概念的不同角度描述**，由于控制反转概念比较含糊（可能只是理解为容器控制对象这一个层面，很难让人想到谁来维护对象关系），所以2004年大师级人物Martin Fowler又给出了一个新的名字：“依赖注入”，相对IoC 而言，“依赖注入”明确描述了“被注入对象依赖IoC容器配置依赖对象”。