# 

**《Spring》**

**Spring IoC**

## 课程主要内容

* 理解IOC
* 基于XML的注入
* Bean的作用域
* 单例问题
* 循环引用问题

目录

[0](#_Toc11088616)

[课程主要内容 0](file:///C:\Users\Administrator\Desktop\课件\Spring\Spring02%20Spring%20IoC.docx#_Toc11088617)

[IoC概念 1](#_Toc11088618)

[IoC 1](#_Toc11088619)

[DI 1](#_Toc11088620)

[基于XML的DI 1](#_Toc11088621)

[XML文件结构 2](#_Toc11088622)

[真的去网上找xsd文件？ 2](#_Toc11088623)

[多配置文件 3](#_Toc11088624)

[Bean的定义与注册 3](#_Toc11088625)

[Alias 别名 3](#_Toc11088626)

[spring ioc container 3](#_Toc11088627)

[构造器注入 constructor-arg 3](#_Toc11088628)

[属性注入 4](#_Toc11088629)

[对其他Bean的引用 5](#_Toc11088630)

[集合 5](#_Toc11088631)

[depends-on 提前初始化 6](#_Toc11088632)

[lazy-init 7](#_Toc11088633)

[作用域 7](#_Toc11088634)

[singleton scope 单例作用域 7](#_Toc11088635)

[prototype scope 原型作用域 7](#_Toc11088636)

[Request scope 7](#_Toc11088637)

[Session scope 7](#_Toc11088638)

[Application scope 7](#_Toc11088639)

[MVC下Spring的单例 7](#_Toc11088640)

[线程安全问题 8](#_Toc11088641)

[成员变量 8](#_Toc11088642)

[引用类型的成员 8](#_Toc11088643)

[controller层 8](#_Toc11088644)

[循环依赖的bean 8](#_Toc11088645)

[构造器注入循环依赖 8](#_Toc11088646)

[通过属性注入 8](#_Toc11088647)

Spring是一个对象管理容器，自动化完成了创建、初始化、销毁等工作。

传统的方式

Person person = **new** Person();

person.setAge(18);

person.setName("zhangsan");

## IoC概念

IoC是一个概念，是一种思想，其实现方式多种多样。当前比较流行的实现方式之一是DI。

# 基于XML的DI

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">

## XML文件结构

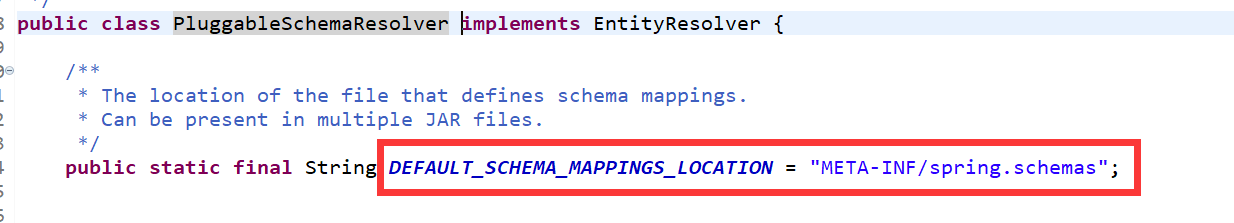
* <beans  beans是xml文件的根节点
* xmlns=[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)xmlns=xml NameSpace 类似于java中的package
* xmlns:xsi=[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)xsi是指xml文件遵守xml规范，xsi全名：xml schema instance
* xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*> 是指具体用到的schema资源

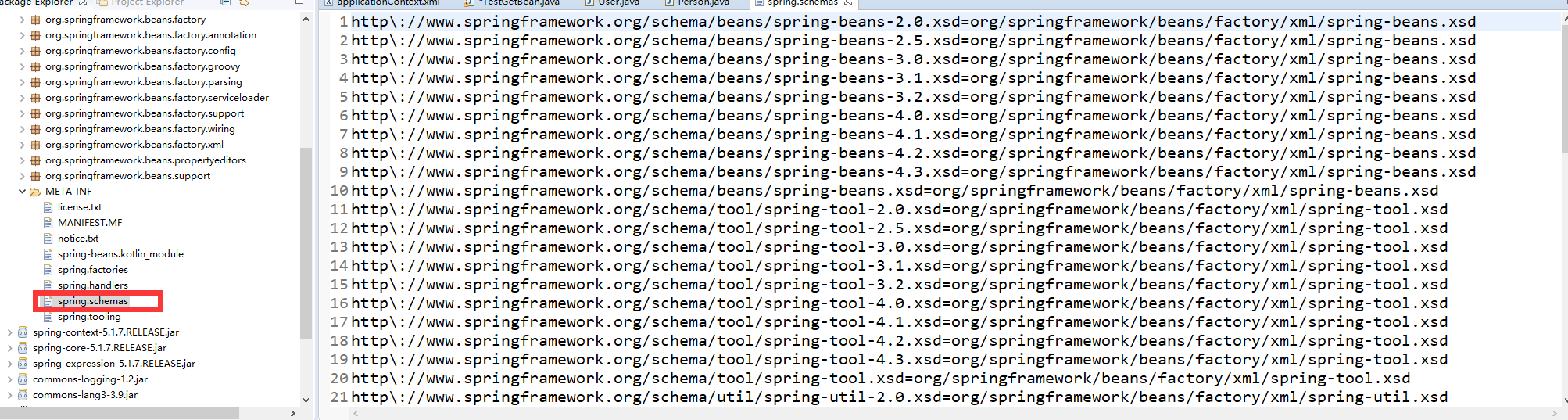
### 真的去网上找xsd文件？

当然不是

spring在加载xsd文件时总是先试图在本地查找xsd文件(spring的jar包中已经包含了所有版本的xsd文件)，如果没有找到，才会转向去URL指定的路径下载

验证PluggableSchemaResolver.class中





=后面是包名以及具体xsd文件位置

### 多配置文件

#### ApplicationContext加载多文件

**new** ClassPathXmlApplicationContext("applicationContext.xml","application-service.xml");

#### 引入外部文件

<import resource="application-service.xml"/>

## Bean的定义与注册

Spring的配置文件是用于指导Spring工厂进行Bean的生产、依赖关系注入及Bean实例分发的“图纸”，它是一个或多个标准的XML文档

<bean id=*"food"* class=*"com.msb.Food"*></bean>

一个bean只能有一个id，但是可以有多个name作为别名

### Alias 别名

<alias name=*"user"* alias=*"my\_user\_bean"* />

## spring ioc container

spring ioc container 管理一个或多个bean，bean来自xml中对bean定义的元数据(configuration metadata)

元数据信息

|  |  |
| --- | --- |
| Class | 类 |
| Name,id | 标识 |
| Scope | 作用域 |
| Constructor arguments | 构造器注入 |
| Properties | 属性注入 |
| autowiring mode | 自动装配 |
| lazy-initialization mode | 懒加载 |
| initialization method | 初始化 |
| destruction method | 销毁 |

### 构造器注入 constructor-arg

Person的构造器

**public** Person(String name, Integer age, Food food) {

**super**();

**this**.name = name;

**this**.age = age;

**this**.food = food;

}

#### Xml

#### 指定name

<bean id=*"person"* class=*"com.msb.Person"*>

<constructor-arg name=*"age"* value=*"18"*></constructor-arg>

<constructor-arg name=*"name"* value=*"zhangsan"*></constructor-arg>

</bean>

#### 指定类型

<constructor-arg type="int" value="7500000"/>

#### 指定index

<constructor-arg index="0" value="7500000"/>

### 属性注入

<bean id="person" class="com.msb.Person">

<property name="age" value="19"></property>

<property name="name" value="zhangsan"></property>

</bean>

#### 使用p-namespace

属性注入

添加一个namespace

xmlns:p=<http://www.springframework.org/schema/p>

使用 p

<bean id="person" class="com.msb.Person" p:age="21" p:name = "zhangsan">

<bean id=*"person"* class=*"com.msb.Person"* p:age=*"21"* p:name = *"zhangsan"* p:food-ref=*"food"*>

#### 使用c- namespace

构造器注入

<bean id="foo" class="x.y.Foo" c:bar-ref="bar" c:baz-ref="baz" c:email= "foo@bar.com"/>

#### 使用java.util.Properties

在set方法中把properties

**private** Properties properties;

**public** **void** setProperties(Properties properties) {

**this**.properties = properties;

**this**.name=properties.getProperty("name");

}

### 对其他Bean的引用

<property name=*"food"* ref=*"food"*></property>

<bean id=*"food"* class=*"com.msb.Food"*></bean>

### 集合

#### Array

private String[] myArray;

<property name="myArray">

<array>

<value>北京</value>

<value>上海</value>

</array>

</property>

#### Properties

**private** Properties adminEmails;

<property name=*"adminEmails"*>

<props>

<prop key=*"administrator"*>administrator@example.org</prop>

<prop key=*"support"*>support@example.org</prop>

<prop key=*"development"*>development@example.org</prop>

</props>

</property>

#### List

**private** List someList;

<property name=*"someList"*>

<list>

<value>apple</value>

<value>orange</value>

</list>

</property>

#### Map

**private** Map someMap;

<property name=*"someMap"*>

<map>

<entry key=*"an entry"* value=*"just some string"*/>

<entry key =*"a ref"* value-ref=*"food"*/>

</map>

</property>

### depends-on 提前初始化

可以使某个bean在创建前，先创建别的bean

### lazy-init

在容器启动后，bean被使用到的时候才加载。可以使用的lazy-init属性

<bean id="person" class="com.msb.Person" lazy-init="false">

## 作用域

spring为bean提供了6种作用域，其中4种只有在web-aware的ApplicationContext种才有用。用户也可以创建自定义的作用域。

singleton 、prototype 、websocket、request、session、application

### singleton scope 单例作用域

每一个类，在一个容器内只能产生一个实例

<bean id="person" class="com.msb.Person" lazy-init="false" scope =”singleton”>

### prototype scope 原型作用域

该bean每次被注入，或者使用getBean()方法获取时，都返回一个新的实例。

<bean id="person" class="com.msb.Person" lazy-init="false" scope =”prototype”>

### Request scope

该作用域的bean，在每个HTTP request都会新建一个实例，当一个request结束后，该实例也会被丢弃。

<bean id="person" class="com.msb.Person" lazy-init="false" scope =”request”>

### Session scope

某一个用户在一段时间内，会使用同一个session，session有超时时间，过了超时时间则session失效。不同用户使用不同的session。

<bean id="person" class="com.msb.Person" lazy-init="false" scope =”session”>

### Application scope

该作用域的bean，每一个application会创建一个

<bean id="person" class="com.msb.Person" lazy-init="false" scope =”application”>

## MVC下Spring的单例

想在一个singleton内多次调用短存活时间的bean（propotype、request、session等），希望调用的是不同的实例，那么就需要使用AOP proxy技术

### 线程安全问题

业务对象并没有做线程的并发限制，因此不会出现各个线程之间的等待问题，或是死锁问题

MVC中的实体bean不是单例的

### 成员变量

在并发访问的时候这些成员变量将会是并发线程中的共享对象，也是影响线程安全的重要因素

### 引用类型的成员

其中引用类型的成员变量即我们在controller中注入的service，在service中注入的dao，这里将其定义为成员变量主

要是为了实例化进而调用里面的业务方法，在这些类中一般不会有全局变量，因此只要我们的业务方法不含有独立的

全局变量即使是被多线程共享，也是线程安全的。

Controller service dao 层中的业务类是多线程共享的，但是每个线程在处理数据的时候具体处理的数据是在每个线程中各自有一份。

### controller层

* final类型 线程安全
* 成员变量 有状态数据有线程安全问题

# 循环依赖的bean

## 构造器注入循环依赖

作业：

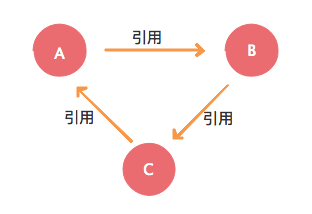
创建3个类

循环引用属性

A类中 -> B类的一引用 -

当循环依赖的bean都是通过构造器注入依赖的时候，无论这些bean是singleton还是prototype，在获取bean的时候都会失败。

## 通过属性注入

* 循环依赖的bean都是singleton 成功，因为单例模式的对象，在Spring中直接实例化；
* 循环依赖的bean都是prototype 失败，因为prototype会检查引用，如果有引用对象没有实例化则暂停实例化，先去实例化它所引用的对象；
* 同时有singleton和prototype 当先获取的那个bean是singleton时，就会成功，否则失败；
* 

当Spring容器在创建A时，会发现其引用了B，从而会先去创建B。同样的，创建B时，会先去创建C，而创建C时，又先去创建A。最后A、B、C之间互相等待，谁都没法创建成功