www.neuedu.com

Neuedu

Spring框架技术
—— 容器和bean的基本原理

V1.1

本章内容

节	知识点	掌握程度	难易程度
IoC介绍	传统组件调用方式	理解	
	IoC	理解	难
	为何使用IoC	理解	
Ioc容器和bean的定义	IoC容器	掌握	
	Simple Object	理解	
	Bean的定义和命名	掌握	
Bean的实例化	Bean的生命周期	理解	
	实例化bean	掌握	
	实例化bean容器	掌握	
	从容器中取得bean	掌握	
	bean的常用属性介绍	掌握	

传统组件调用方式

Neuedu

```
public class Business
{
    private UserDao dao=new UserDao();
    public List getOnPage()
    {
        return this.dao.query();
    }

}
```

传统依赖关系

缺陷分析

Neuedu

- 无法真正动态加载
  - 无法动态替换。替换时必须修改代码，即使应用的是工厂，依然存在代码修改的情况，因为我们必须重新设计工厂。

- 实例无法共享
  - 多组件无法共享同一实例。因为实例的生命周期是依赖于调用者组件，二者同生共死，即使应用工厂，也是每个实例只为一个客户端使用。

- 测试困难
  - 因为必须包含一个组件的实例，如果实例包含了数据库引用的话，无法离开数据库运行环境。

对象创建方式

- 原始社会
  - UserBusines business=new UserBusiness();
- 封建社会
  - 程序世界进入半现代社会，我们应用工厂进行解耦，虽然使用者不再new一个对象了，可是，也不过仅仅是把对象的创建过程封装到了工厂中而已，我们毕竟还是new了一个对象。耦合并没有完全解除。
- 示例 spring-whySpring-1、spring-whySpring-2工程

开发人员面临的窘境

- 代码侵入性、依赖性较强
- 单元测试极其困难
- 代码混乱，维护性和扩展性差
- 代码难以复用
- 不利于分工，降低开发效率

不使用Spring的代码

Neuedu

- 没有使用Spring：
  - 代码中充斥了很多工厂类、singleton单例模式；
  - 配置也不够集中，没有一个统一的管理；
  - 在业务层，我们一般都需要依赖Dao，我们需要自己写一些工厂类来生成；
  - 通过示例可见这个组装过程是由我们来做的，而且我们已经把这个写死了，对象是我们new出来的。在程序里，这样的对象很多，对于一个大型的项目，不能这样都写死了！而且这样装配会很复杂，如果Dao还依赖别的东西的话，还要继续装配。
- 示例：示例 spring-whySpring-1、spring-whySpring-2工程

使用IoC解耦

Neuedu

- 解决之道就是应用IoC
  - 让组件之间的依赖关系通过抽象（接口或抽象类的变量）来建立。这样，在组件运行期间，将组件依赖的实际对象注入（填充）进来，并由组件内部包含的抽象变量来引用，就可以解藕了。

IoC（控制反转）

- IoC的直译是控制反转。

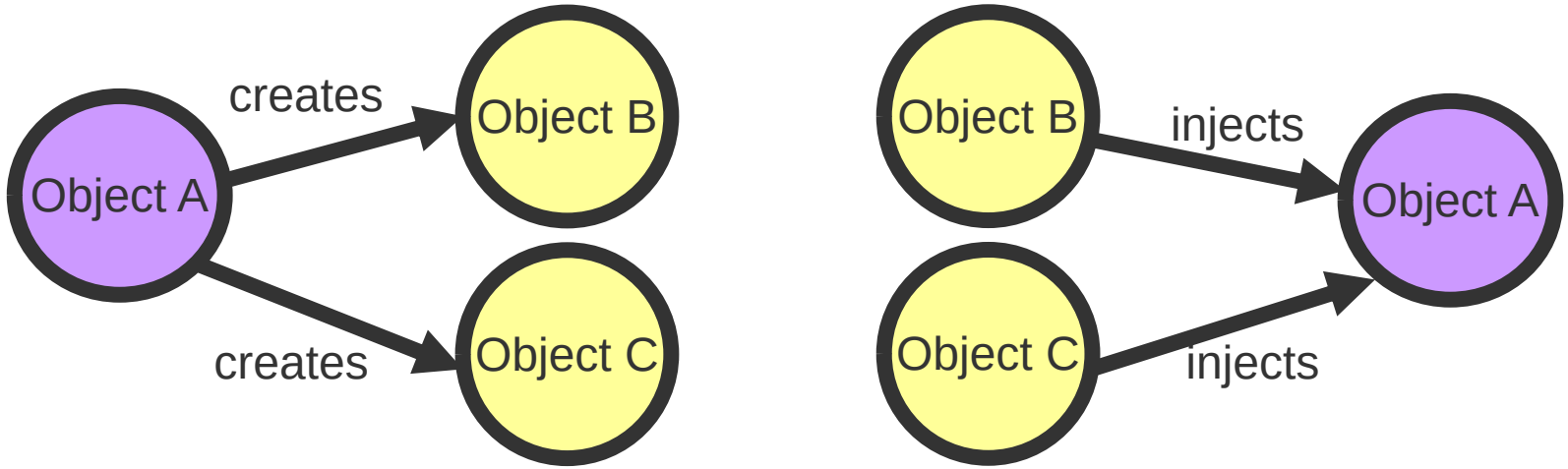
- 在IoC模式下，控制权限从应用程序转移到了IoC容器中。组件不是由应用程序负责创建和配置，而是由IoC容器负责。
- 使用IoC的情况下，对象是被动地接收依赖类而不是主动地查找。对象不是从容器中查找他的依赖类，而是容器在实例化对象时，主动地将他所依赖的对象注入给他。
- 应用程序只需要直接使用已经**创建并且配置好**的组件即可，而不必自己负责创建和配置。

为何使用IoC

- 有效地组织中间层对象
- 有效的消除单例、工厂等模式的使用
- 将面向接口编程做到实处
- 使单元测试变得简单
- 消除了依赖环境的查找和特定服务器的代码
- 代码变得清晰，易于维护和扩展
- 代码依赖于接口，易于复用
- 便于看清组件之间的依赖关系
- 便于团队分模块开发

Without IoC vs. With IoC

Neuedu



依赖方式比较

Neuedu

- 未使用IoC

```
class A{
    B b = new
B();
}
```

管理

```
class B{
    C c = new
C();
}
```

管理

```
classC{
    D d = new
D();
}
```

- 使用IoC

```
class A{
    B b;
    void setB(B
b)
    { this.b =
b; }
}
```

注入

```
class B{
    C c;
    void setC(C
c)
    { this.c =
c; }
}
```

注入

```
class C{
    D d;
    void setD(D
d)
    { this.d =
d; }
}
```

IOC容器

- 何谓Ioc:由容器来管理对象之间的关系（而不是对象本身来管理），就是控制反转或是依赖注入。
- Spring框架的基本思想就是控制反转、依赖注入。
- Spring就是一个Ioc容器。

- 示例：示例 spring-whySpring-1、spring-whySpring-2工程

IoC容器—BeanFactory

- 轻量级的IoC容器
  - org.springframework.beans.factory.BeanFactory
- 管理Bean的生命周期及其依赖关系
- 尽可能晚的初始化Bean，适用于对内存要求很高的应用
- 最常用的是XmlBeanFactory

  e.g.：BeanFactory factory =new XmlBeanFactory(new FileInputStream("beans.xml"));
- key API
  - boolean containsBean(String)
  - Object getBean(String)
  - boolean isSingleton(String)

IoC容器— ApplicationContext

- BeanFactory功能的延伸
- 尽可能早的初始化Bean
- 支持国际化
- 支持事件监听机制
- 提供通用的方式获取资源
- 主要实现
  - ClassPathXmlApplicationContext
  - FileSystemXmlApplicationContext
  - XmlWebApplicationContext

Neuedu

ApplicationContext与BeanFactory

- BeanFactory
  - 采用延迟加载Bean，直到第一次使用getBean()方法获取Bean实例时，才会创建Bean。
- ApplicationContext
  - ApplicationContext在自身被实例化时一次完成所有Bean的创建。大多数时候使用ApplicationContext。

- 区别
  - 在服务启动时ApplicationContext就会校验XML文件的正确性，不会产生运行时bean装配错误。
  - BeanFactory在服务启动时，不会校验XML文件的正确性，获取bean时，如果装配错误马上就会产生异常。

Simple Object

Neuedu

- Simple Object范围：
  - POJO：简单的Java对象（Plain Old Java Objects）实际就是普通JavaBean
  - VO：value object值对象
  - PO：persistant object 持久对象
  - DTO：Data Transfer Object 数据传输对象
  - DAO：data access object 数据访问对象

- 范例
  - Simple Object在不使用事务API的情况下能够处理事务
  - Simple Object能够在不依赖于任何接口的具体实现情况下完成业务操作
  - Simple Object不需实现任何接口即可发布成web服务供远程调用

Bean是什么

- 具有唯一id的Simple Object
- 由IoC容器**管理其生命周期及其依赖关系**
  - 简单地讲，bean就是由Spring容器初始化、装配及被管理的对象
  - bean定义以及bean相互之间的依赖关系将通过配置元数据来描述
- 一般在XML文件中定义

Bean 的定义和命名

- Spring IoC容器至少包含一个bean定义，但大多数情况下会有多个bean定义。当使用基于XML的配置元数据时，将在顶层的<beans/>元素中配置一个或多个<bean/>元素。
- bean定义与应用程序中实际使用的对象一一对应。通常情况下bean的定义包括：
  - 服务层对象
  - 数据访问层对象（DAO）
  - 类似Struts Action的表示层对象
  - Hibernate SessionFactory对象等等。

项目的复杂程度将决定bean定义的多寡。

Bean 的定义和命名

- 以下是一个基于XML的配置元数据的基本结构：

```
<?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-2.5.xsd">

    <bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
    </bean>
    <bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
    </bean> <!-- more bean definitions go here... -->

</beans>
```

示例： spring-whySpring-2工程

Bean 的定义和命名

Neuedu

```
<bean id="personService"
class="com.neusoft.beans.PersonServiceBean"
lazy-init="false"
init-method="init"
destroy-method="destory"
scope="prototype">

</bean>
```

Bean 的定义和命名

- Spring IoC容器将管理一个或多个*bean*，这些bean将通过配置文件中的bean定义被创建(在XML格式中为<bean/>元素)。
- 在容器内部，这些bean定义由BeanDefinition 对象来表示，该定义将包含以下信息：
  - *全限定类名：*这通常就是已定义bean的实际实现类。
  - bean行为的定义，即创建模式（prototype还是singleton）、自动装配模式、依赖检查模式、初始化以及销毁方法。这些定义将决定bean在容器中的行为
  - 用于创建bean实例的构造器参数及属性值。bean之间的关系，即*协作*（或者称依赖）。

Bean 的定义和命名

属性名称	属性描述
class	指定实例化对象的类型，class 属性通常是必须的。
name	通过id或name属性来指定bean标识符，**通常情况下最好为bean指定一个id，id属性具有唯一性。**
scope	定义该bean的作用域（有5种作用域）
constructor arguments	使用构造器参数来注入依赖关系
autowiring mode	可以*指定自动装配（autowire）*相互协作bean之间的关联关系 （5种类型）
dependency checking mode	对容器中bean的依赖设置进行检查 ，可以检查bean定义中实际属性值的设置
lazy-initialization mode	如果你不想让一个singleton bean在ApplicationContext实现在初始化时被提前实例化，那么可以将bean设置为延迟实例化。一个延迟初始化bean将告诉IoC 容器是在启动时还是在第一次被用到时实例化。在XML配置文件中，延迟初始化将通过<bean/>元素中的lazy-init属性来进行控制”true/false”.
initialization method	不做要求
destruction method	不做要求

Bean的生命周期管理

Neuedu

- IoC容器管理的JavaBean的生命周期划分为四个阶段
  - 实例化JavaBean
  - JavaBean实例的初始化，即通过IoC注入其依赖性
  - 基于Spring应用对JavaBean实例的使用
  - IoC容器销毁JavaBean实例

- 示例： spring-life工程

实例化bean

Neuedu

- 就Spring IoC容器而言，bean定义基本上描述了创建一个或多个实际bean对象的内容。当需要的时候，容器会从bean定义列表中取得一个指定的bean定义，并根据bean定义里面的配置元数据，使用反射机制来创建一个实际的对象。因此需要告知Spring IoC容器我们将要实例化的对象的类型以及如何实例化对象。
  - 用构造器来实例化
  - 使用 静态工厂方法实例化
  - 使用实例工厂方法实例化
- 示例：spring-instance工程

实例化bean--1

Neuedu

- 通过构造方法直接创建：
  <bean id=“..” class=“..”>

```
<bean id="exampleBean" class="examples.ExampleBean"/>
```

实例化bean--2

Neuedu

- 通过静态工厂方法创建：
  <bean id="..." class="..." factory-method="...">

- <bean id="exampleBean" class="examples.ExampleBean2"
  factory-method="createInstance"/>

- package examples;
  public class ExampleBean2 {
  public static Mybean createInstance() {
  return new Mybean();
  }
  }

实例化bean--3

- 通过实例工厂方法（非静态）创建：

```
<bean id="factory" class="..">
<bean id=".." factory-bean="factory" factory-method="..">
<!-- the factory bean, which contains a method called createInstance() -->
<bean id="myFactoryBean" class=" examples.MyFactoryBean"> ... </bean>
<!-- the bean to be created via the factory bean -->
<bean id="exampleBean" factory-bean="myFactoryBean"
      factory-method="createInstance"/>
```

-
```
package examples;
public class MyFactoryBean{
      public Mybean createInstance() {
            return new Mybean();
       }
}
```

实例化Ioc容器

Neuedu

- 实例化Ioc容器常用的两种方式：
  - 在类路径下寻找配置文件来实例化容器
  - 在文件系统路径下寻找配置文件来实例化容器

```
ApplicationContext ctx = new ClassPathXmlApplicationContext(new
    String[]{"beans.xml"});
ApplicationContext ctx = new FileSystemXmlApplicationContext(new
    String[]{“d:\\beans.xml“});
```

Spring的配置文件可以指定多个，可以通过String数组传入。

从Ioc容器中得到bean

Neuedu

- 从本质上讲，BeanFactory仅仅只是一个维护bean定义以及相互依赖关系的高级工厂接口。通过使用getBean(String) 方法就可以取得bean 的实例；
- ApplicationContext ctx = new
  ClassPathXmlApplicationContext(“beans.xml”);
- OrderService service =
  (OrderService)ctx.getBean("personService");

Bean的作用域

Neuedu

. singleton

在每个Spring IoC容器中一个bean定义只有一个对象实例。

默认情况下会在容器启动时初始化bean，但我们可以指定Bean节点的lazy-init=“true”来延迟初始化bean，这时候，只有第一次获取bean会才初始化bean。

如：

```
<bean id="xxx" class="cn. neusoft.OrderServiceBean" lazy-init="true"/>
```

如果想对所有bean都应用延迟初始化，可以在根节点beans设置default-lazy-init=“true“，如下：

```
<beans default-lazy-init="true“ ...>
```

. prototype

每次从容器获取bean都是新的对象。

. request

. session

. global session

示例： spring-scope、spring-life工程

Neuedu

指定Bean的初始化方法和销毁方法

- 指定Bean的初始化方法和销毁方法

```
<bean id="xxx" class="cn. neusoft.OrderServiceBean" init-method="init" destroy-method="close"/>
```

示例： spring-life工程

作用域	描述
singleton	在每个Spring IoC容器中一个bean定义对应一个对象实例。
prototype	一个bean定义对应多个对象实例。
request	在一次HTTP请求中，一个bean定义对应一个实例；即每次HTTP请求将会有各自的bean实例，它们依据某个bean定义创建而成。该作用域仅在基于web的Spring ApplicationContext情形下有效。
session	在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。
global session	在一个全局的HTTP Session中，一个bean定义对应一个实例。典型情况下，仅在使用portlet context的时候有效。该作用域仅在基于web的Spring ApplicationContext情形下有效。

bean的作用域-初始化web配置

- 要使用request、session和 global session作用域的bean（即具有web作用域的bean），在开始设置bean定义之前，还要做少量的初始配置。singleton和prototype，就不需要这一额外的设置。
- 在目前的情况下，根据你的特定servlet环境，有多种方法来完成这一初始设置。如果你使用的是Servlet 2.4及以上的web容器，那么你仅需要在web应用的XML声明文件web.xml中增加下述ContextListener即可：

bean的作用域-初始化web配置

- <web-app>
  ...
- <listener>
- <listener-class>
  org.springframework.web.context.request.
  RequestContextListener
- </listener-class>
- </listener>
  ...
- </web-app>

bean的作用域-初始化web配置

- Request作用域
  考虑下面bean定义：

```
<bean id="loginAction" class="com.foo.LoginAction"
        scope="request"/>
```

  针对每次HTTP请求，Spring容器会根据loginAction bean定义创建一个全新的LoginAction bean实例，且该loginAction bean实例仅在当前HTTP request内有效，因此可以根据需要放心的更改所建实例的内部状态，而其他请求中根据loginAction bean定义创建的实例，将不会看到这些特定于某个请求的状态变化。当处理请求结束，request作用域的bean实例将被销毁。

bean的作用域-初始化web配置

- Session作用域

  考虑下面bean定义：

  ```
  <bean id="userPreferences" class="com.foo.UserPreferences"
        scope="session"/>
  ```

  针对某个HTTP Session，Spring容器会根据userPreferences bean定义创建一个全新的userPreferences bean实例，且该userPreferences bean仅在当前HTTP Session内有效。与request作用域一样，你可以根据需要放心的更改所创建实例的内部状态，而别的HTTP Session中根据userPreferences创建的实例，将不会看到这些特定于某个HTTP Session的状态变化。当HTTP Session最终被废弃的时候，在该HTTP Session作用域内的bean也会被废弃掉。

bean的依赖（depends-on）

Neuedu

- 一个bean对另一个bean的依赖最简单的做法就是将一个bean设置为另外一个bean的属性。
- 在xml配置文件中最常见的就是使用<ref/>元素。
- 被依赖bean将在依赖bean之前被适当的初始化。
- depends-on属性可以用于当前bean初始化之前显式地强制一个或多个bean被初始化。
- 若需要表达对多个bean的依赖，可以在‘depends-on’中将指定的多个bean名字用分隔符进行分隔，分隔符可以是**逗号、空格及分号**等。

```
<bean id="beanOne" class="ExampleBean"
                                depends on="manager,accountDao">
    <property name="manager" ref="manager" />
</bean>
<bean id="manager" class="ManagerBean" />
<bean id="accountDao" class="x.y.jdbc.JdbcAccountDao" />
```

Bean的延迟初始化（lazy-init）

Neuedu

- singleton bean的实例化有两种方式：
  - 默认行为就是在ioc容器启动时将所有singleton bean提前进行实例化；
  - singleton bean在第一次使用时被实例化

- 在XML配置文件中，延迟初始化将通过<bean/>元素中的lazy-init属性来进行控制。
- 例如：

```
<bean id="lazy" class="com.foo.ExpensiveToCreateBean"  lazy-init="true">
    <!-- various properties here... -->
</bean>
<bean name="not.lazy" class="com.foo.AnotherBean">
    <!-- various properties here... -->
</bean>
```

Neuedu

本章重点总结

- 了解IOC容器和bean的概念
- 掌握bean的定义、配置项、实例化
- 掌握容器的实例化

Neuedu