# J2EE

###### 1.绪论

Maven项目结构：src项目资源的总目录，所有操作都在这个文件夹执行

Main：项目的主目录

webAPP：前端部分，包含前端页面，css，js等文件

WEB-INF：java的WEB应用安全目录，客户端无法直接访问，只有服务器端可以直接访问。如果页面想要访问这个目录必须通过web.xml进行映射

Web.xml：沟通前后端的桥梁，配置网站入口，声明哪些文件是后端配置文件

Pom.xml：主要描述maven坐标，包依赖关系

External Libraries：从字面就可以了解，就是外部的一些包。也就是通过Maven下载的包，JDK等等。

###### Spring

IOC：由容器来负责控制对象的生命周期和对象间的关系。控制反转，将设置好的类交给系统去控制

DI：依赖注入。容器动态的将某种依赖关系注入到组件之中。目的是为了提示组件的重用率。

DI的两种方式：

设值注入：使用IOC容器的setter方法注入。<property name=”userDao” ref=”userDao” />.

构造注入：使用构造器来注入被依赖的实例。<constructor-arg ref=”db” />.

设值注入的优点：

1. 设值注入的javabean的写法与传统写法一致，更容易被接受。
2. 对于复杂的依赖关系，如果采用构造注入会导致构造器过于臃肿，spring在创建bean对象的同时要依赖化其他所有实例化对象，导致性能下降。
3. 对于某些属性可选的情况，多参数构造器更复杂。

构造注入的优点：

1. 如果对依赖注入的顺序有要求则可以在构造器种设定依赖注入的顺序
2. 对依赖关系无需变化bean
3. 使用构造器注入，组件内部的依赖关系完全对组件调用者完全透明，更符合高内聚原则。

一般以设值注入为主，构造注入为辅。

Bean的配置管理：

Spring中bean的定义：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.springframework.org/schema/beans"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- 配置stoneAxe实例，其实现类是StoneAxe -->

<bean id="stoneAxe" class="org.crazyit.app.service.impl.StoneAxe"/>

<!-- 配置steelAxe实例，其实现类是SteelAxe -->

<bean id="steelAxe" class="org.crazyit.app.service.impl.SteelAxe"/>

</beans>

配置依赖：各组件的相互引用实质就是依赖关系，可以通过spring的IOC容器完成注入。依赖关系可以是一个确定的常亮也可以是对其他bean的引用，通常普通属性不纳入配置管理，主要是配置bean实例之间依赖的关系。

注入普通属性：

<bean id="exampleBean" class="org.crazyit.app.service.ExampleBean">

<!-- 确定int型属性值 -->

<property name="integerProperty" value="1"/>

<!-- 确定double型属性值 -->

<property name="doubleProperty" value="2.3"/>

</bean>

注入bean引用：

<bean id="chinese" class="org.crazyit.app.service.impl.Chinese">

<!-- 将stoneAxe注入给axe属性 -->

<property name=“stoneAxe" ref="stoneAxe"/>

</bean>

<!-- 配置stoneAxe实例，其实现类是StoneAxe -->

<bean id="stoneAxe" class="org.crazyit.app.service.impl.StoneAxe"/>

使用自动装配完成bean的引用：设值bean的autowrie属性

<bean id="auto" class="example.autoBean" autowire="byType"/>

no：不使用自动装配

Byname：根据名字自动装配。在容器中找到id属性与需要注入的属性（setter方法的名字）同名的bean完成注入。

Bytype：根据类型自动匹配

constructor：与byType类似

autodetect：根据bean内部结构决定用constuctor还是byType

注入嵌套bean：

<bean id="exampleBean" class="org.crazyit.app.service.ExampleBean">

<property name="axe">

<bean class="org.crazyit.app.service.impl.StoneAxe"/>

/property>

</bean>

创建bean的实例：

<bean id="chinese" class="org.crazyit.app.service.impl.Chinese">

<property name="axe" ref="steelAxe"/>

</bean>

创建：

ApplicationContext ctx = new ClassPathXmlApplicationContext("bean.xml");

Person p = (Person)ctx.getBean("chinese");

//Person p = ctx.getBean("chinese" , Person.class);

p.useAxe();

Bean的作用域：

Singleton：单例模式，整个容器中只有一个实例

Prototype:原型模式，每次通过getbean获取时创建一个实例

Request：每次http请求产生一个新的实例，需要在web.xml中配置listener

Session：每次http session产生一个新的实例

Bean的生命周期：

Prototype bean每次请求都会产生一个新的实例对象，spring无法跟踪其生命周期。

singleton bean：单例对象，每次客户请求都返回同一个共享实例，spring可以跟踪其创建和销毁，实现对其生命周期的管理

生命周期的管理时机：

注入之后---init method

在bean的配置中加入inti-method属性指定全部属性注入/设值成功后要执行的方法

<bean id="chinese" class="org.j2ee.service.impl.Chinese"

init-method="init">

<property name="axe" ref="steelAxe"/>

</bean>

Bean销毁之前

占用资源的回收

在<bean>的配置中加入destroy-method属性指定bean在销毁之前要执行的方法

<bean id="chinese" class="org.j2ee.service.impl.Chinese"

destroy-method=“destroy">

<property name="axe" ref="steelAxe"/>

</bean>

Singleton bean依赖Prototype bean作用域不同步，原因是Singleton bean只会创建一次之后容器内共享，而Prototype bean每次请求都会创建一个新的实例对象

解决方案

放弃容器注入，singleton bean每次使用prototype bean时主动从服务器请求新的prototype bean。耦合了SpringAPI

利用方法注入：使用lookup方法注入