# Spring学习笔记

# 一. IoC容器

1. spring 能够让你用普通java对象(POJO)来构建应用程序
2. 模块:

**Core Container: 核心容器,** 包含Beans, Core, Context, EL。

Beans和Core提供了IoC和DI功能;

Context模块继承Beans模块的特性，并增加对国际化（例如使用资源包），事件传播，资源加载以及通过例如servlet容器进行上下文透明创建的支持。ApplicationContext接口是Context模的核心;

EL模块提供了强大的表达式语法功能, 的表达式语言模块提供用于查询并在运行时操作对象图的强大的表达式语言。该语言支持设置和获取属性值，属性赋值，方法调用，访问数组的上下文，集合和索引器，逻辑和算术运算符，命名变量以及从Spring的IoC容器中按名称检索对象。它还支持列表预测和选择以及常用列表聚合。

**数据访问和集成模块**: JDBC, ORM, OXM, JMS and Transaction modules.

**WEB:** web, struts, servlet, portlet.

**AOP:**

**TEST:**

1. bean: 由IoC容器管理的对象称为bean.
2. 配置元数据 + 业务POJO ====🡺 可执行的应用程序.

5. Spring元数据配置方式:

5.1 xml格式.

5.2 基于注解的配置: Spring 2.5

5.3 基于Java的配置.Spring 3.0, 参阅@Configuration，@Bean，@Import 和@DependsOn注释。

1. Xml 配置:

将bean 在顶层元素<beans>中配置<bean/>

1. Xml文件可以根据不同的逻辑模块分类定义, 然后通过<beans/>元素的子元素<import resource=””/>引入.
2. <beans/>元素

<beans>

<bean/> 定义bean

<import/> 引入xml配置文件

<alias name="" alias=""/> 给<bean>定义别名

<description> 当前文件的描述性文档

</beans>

1. <bean/>元素, 该元素的各项配置在Spring中最终都是转为**BeanDefinition**对象.bean元素的各项属性可以在AbstractBeanDefinition中查看.

<bean/>元素的属性:

class: 全限定类名

id: bean的标识符,在容器内具有唯一值.

name: bean的标识符, 可以定义多个, 用逗号,分号,空格分隔. 在容器内具有唯一值.

scope: 作用域, 默认单例, 原型prototype. 如果有父bean的话,会继承父bean的作用域.

lazy-init: 延迟加载.

init-method: 初始化方法

destory-method: 销毁方法.

abstract: 定义当前bean是否是一个抽象bean, 当前bean不能被实例化, 只能被继承. 默认为false. true.

parent: 指定父bean.

autowire: 自动装配, 默认为no, byName, byType.

factory-bean:

factory-method:

depends-on: 指定依赖

1. 实例化bean, bean定义本质上是创建一个或多个对象的配方.
   1. 嵌套的static内部类的bean定义. 如类Foo中有一个静态类Bar, 则class属性为: com.example.Foo**$**Bar.
   2. 构造函数实例化, bean要有默认构造方法. <bean id=”” class=””/>
   3. **静态工厂方法**实例化bean, bean类要有静态方法(作为实例化工厂方法)

<bean id = “student” class = “examples.Student” factory-method = “newInstance” />

public class Student {

private static Student student = new Student();

private Student(){}

public static Student newInstance(){  
 return student;

}

}

* 1. 实例工厂方法实例化bean. **从容器调用现有的bean的非静态方法来创建新的bean.**此时class属性留空.

<bean id = “student” class = “test.Student”/>

<bean id = “student” factory-bean = “student” factory-method = “getAddress” />

public class Student {

private Address address = new Address();

private Student(){}

public Address getAddress(){  
 return address;

}

}

public class Address {

private String city;

…… getter setter

}

一个工厂类可以有多个工厂方法.

1. 依赖注入: 构造器注入 和 setter注入
   1. 构造器注入: Spring推荐使用构造器注入, 这样可以实现不可变类(没有setter方法). 注意: 构造器注入无法解决循环注入的场景(类A的构造函数中调用类B, 类B的构造函数中调用类A),此时要用setter注入.

**package** x.y;

**public** **class** Foo {

**public** Foo(Bar bar, Baz baz) {

*// ...*

}

}

如果此时Bar与Baz与继承无关,且不存在潜在的歧义,在注入时无需指定索引或类型

<beans>

<bean id="foo" class="x.y.Foo">

<constructor-arg ref="bar"/>

<constructor-arg ref="baz"/>

</bean>

<bean id="bar" class="x.y.Bar"/>

<bean id="baz" class="x.y.Baz"/>

</beans>

**package** examples;

**public** **class** ExampleBean {

**private** **int** years;

**private** String ultimateAnswer;

**public** ExampleBean(**int** years, String ultimateAnswer) {

**this**.years = years;

**this**.ultimateAnswer = ultimateAnswer;

}

}

1. 当Spring无法确定value的类型时,则可指定**参数类型**

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

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

<constructor-arg type="java.lang.String" value="42"/>

</bean>

1. 也可通过指定**参数顺序**来注入,索引从0开始

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

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

<constructor-arg index="1" value="42"/>

</bean>

1. 也可通过指定**参数名称**来注入

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

<constructor-arg name="years" value="7500000"/>

<constructor-arg name="ultimateAnswer" value="42"/>

</bean>

* 1. setter注入: 调用bean的setter方法注入

**public** **class** ExampleBean {

**private** AnotherBean beanOne;

**private** YetAnotherBean beanTwo;

**private** **int** i;

**public** **void** setBeanOne(AnotherBean beanOne) {

**this**.beanOne = beanOne;

}

**public** **void** setBeanTwo(YetAnotherBean beanTwo) {

**this**.beanTwo = beanTwo;

}

**public** **void** setIntegerProperty(**int** i) {

**this**.i = i;

}

}

setter注入配置:

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

*<!-- setter injection using the nested ref element -->*

<property name="beanOne">

<ref bean="anotherExampleBean"/>

</property>

*<!-- setter injection using the neater ref attribute -->*

<property name="beanTwo" ref="yetAnotherBean"/>

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

</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>

<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>

* 1. 依赖注入配置详解

1. 字符串, 基本类型, spring会自动转为相应的类型

<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

*<!-- results in a setDriverClassName(String) call -->*

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/mydb"/>

<property name="username" value="root"/>

<property name="password" value="masterkaoli"/>

</bean>

1. 使用p-namespace

<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="close"

p:driverClassName="com.mysql.jdbc.Driver"

p:url="jdbc:mysql://localhost:3306/mydb"

p:username="root"

p:password="masterkaoli"/>

1. ref : 当引用一个bean时,则用ref=”” 或<ref bean=””/>
2. 内部bean, <bean/>可以位于<property/>或<constructor-arg/>内部

<bean id="outer" class="...">

*<!-- instead of using a reference to a target bean, simply define the target bean inline -->*

<property name="target">

<bean class="com.example.Person"> *<!-- this is the inner bean -->*

<property name="name" value="Fiona Apple"/>

<property name="age" value="25"/>

</bean>

</property>

</bean>

注意:内部bean是匿名的,无需为其指定id或name,无法通过id或name直接进行访问.

1. 注入集合 <list/> <set/> <map/> <props/>

<bean id=”” class=””>

<property name=””>

<!—注入properties -->

<props>

<prop key=”url”>https://xxxxx</prop>

</props>

</property>

<property name=””>

<!-- 注入list集合 list与set格式 相同-->

<list>

<value>30</value>

<ref bean=””/>

</list>

</property>

<property name=””>

<!-- 注入map集合 -->

<map>

<entry key=”an entry” value=”a string”/>

<entry key=”a ref” value-ref=”a bean”/>

</map>

</property>

</bean>

1. 集合合并

<list/> <map/> <set/> <props/>可以通过继承来合并.此时要用到以上各元素的【merge=”true”】属性.

当merge=true时会将父子的值合并(相同则以子为准),当merge=false时则是子值代替父值(若子值未定义,则为父值),list会保证元素的顺序(先父后子),其他则无序.

merge属性只能在子级中指定.

不能合并不同类型的集合.

<beans>

<bean id="parent" abstract="true" class="example.ComplexObject">

<property name="adminEmails">

<props>

<prop key="administrator">administrator@example.com</prop>

<prop key="support">support@example.com</prop>

</props>

</property>

</bean>

<bean id="child" parent="parent">

<property name="adminEmails">

*<!-- the merge is specified on the child collection definition -->*

<props merge="true">

<prop key="sales">sales@example.com</prop>

<prop key="support">support@example.co.uk</prop>

</props>

</property>

</bean>

<beans>

1. Null和空字符值注入

<bean class="ExampleBean">

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

</bean>

以上等价于: exampleBean.setEmail("");

<bean class="ExampleBean">

<property name="email">

<null/>

</property>

</bean>

以上等价于: exampleBean.setEmail(null);

1. p-namespace:指<property/>元素可以通过<bean/>元素中设置p:name形式定义.

<bean name="john-classic" class="com.example.Person">

<property name="name" value="John Doe"/>

<property name="spouse" ref="jane"/>

</bean>

等价于:

<bean name="john-modern"

class="com.example.Person"

p:name="John Doe"

p:spouse-ref="jane"/>

<bean name="jane" class="com.example.Person">

<property name="name" value="Jane Doe"/>

</bean>

此种形式的局限性为属性名(以ref结尾)与ref结构的冲突.

1. c-namespace:用于替代<constructor-arg/>元素

<bean id="bar" class="x.y.Bar"/>

<bean id="baz" class="x.y.Baz"/>

*<!-- traditional declaration -->*

<bean id="foo" class="x.y.Foo">

<constructor-arg ref="bar"/>

<constructor-arg ref="baz"/>

<constructor-arg value="foo@bar.com"/>

</bean>

*<!-- c-namespace declaration -->*

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

对于某些构造函数名不可用的特殊情况,可以用索引.

<bean id="foo" class="x.y.Foo" c:\_0-ref="bar" c:\_1-ref="baz"/>

10.复合属性, 可以用复合属性设置bean的属性.

<bean id="foo" class="foo.Bar">

<property name="fred.bob.sammy" value="123" />

</bean>

注意:fred和bob都不可为空,否则会抛出异常