







# 1、搭建Spring的IOC开发环境

第一步：拷贝必备jar包到lib目录

context,core,beans,expression-language

两个依赖jar包：都是跟日志相关的

 commons-logging-1.2.jar	2016/12/22 18:58	Executable Jar File	61 KB
 log4j-1.2.16.jar	2016/12/22 18:59	Executable Jar File	471 KB
 spring-beans-4.2.4.RELEASE.jar	2016/12/22 18:59	Executable Jar File	715 KB
 spring-context-4.2.4.RELEASE.jar	2016/12/22 18:58	Executable Jar File	1,072 KB
 spring-core-4.2.4.RELEASE.jar	2016/12/22 18:58	Executable Jar File	1,054 KB
 spring-expression-4.2.4.RELEASE.jar	2016/12/22 18:58	Executable Jar File	257 KB

第二步：在类的根路径下创建一个任意名称的xml的配置文件。

需要导入约束

约束文件：[spring-framework-4.2.4.RELEASE\docs\spring-framework-reference\html\xsd-configuration.html](#)

## IOC的基础约束

```
<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">

</beans>
```

## 2、入门案例

spring的ioc最基本使用

2.1、写配置，把需要new出来的资源交给spring来管理

```

<!-- 把资源交给spring来管理
    id:任意写但一个配置文件中不能有相同的ID
    class: 资源文件全路径 -->
<bean id="Jdao" class="com.dao.impls.Jdao"></bean>

<!-- get注入要注意 property name="SDdao" 这个name的名字要与get注入的get方法名致 否则会报错
    property: 配置资源中的资源归属
    name: 资源的get方法名
    ref: 指向的资源 -->
<bean id="JdbcService" class="com.services.JdbcServices">
    <property name="SDdao" ref="Jdao"></property>
</bean>

```

2.2、在需要资源的类中通过配置文件获取到spring的容器，通过ID获取资源对象

**ClassPathXmlApplicationContext:** 它是只能在类路径下读取spring的配置文件，不在类路径下，读取不到。在java工程中用这种方式。

**FileSystemXmlApplicationContext:** 它是可以在文件系统中读取spring的配置文件，只要在文件系统中，就都可以读取到。

**BeanFactory:** 它是spring提供的工厂。可以从容器中根据id获取到一个对象。它提供的是延迟加载的策略。什么时候用，什么时候创建对象。

**ApplicationContext:** 它是spring提供工厂的一个子接口。也可以从容器中根据id获取到一个对象。它提供的是立即加载的策略。实际开发中用这种

\* 不管用不用，只要一读取配置文件，就创建对象。

```

//1. 通过配置文件获取到spring的容器
ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
//2. 通过ID获取对象
ICustomerService customerService = (ICustomerService) ac.getBean("customerService");

```

1.通过配置文件获取到spring的容器

```
ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");
```

2.通过ID获取对象

```
ICustomerService customerService = (ICustomerService) ac.getBean("customerService");
```

### 3、bean实例化，管理工厂

bean的实例化：(默认情况下)

spring会调用bean的默认**无参构造函数来实例化**。如果没有默认无参构造函数，则不能创建成功。

spring管理静态工厂

```
<bean id="staticFactory" class="com.itheima.spring.StaticFactory" factory-
```

```
method="getUserBean"></bean>
```

spring管理实例工厂

```
<bean id="instancFactory" class="com.itheima.spring.InstanceFactory"></bean>
```

```
<bean id="instanceFactoryBean" factory-bean="instancFactory" factory-  
method="getUserBean"></bean>
```

## 4、bean在项目中的作用范围

bean的作用范围：

单例的：作用范围是整个应用 spring的bean默认都是单例的。

spring支持配置作用范围的方式：

scope：指定bean的作用范围

singleton：单例的（默认值）

**prototype**：多例的。在整合struts2框架之后，action的配置必须采用此值

=====以上两个必须掌握，以下3个了解即可

=====

request：作用范围是一次请求和当前请求的转发。

session：作用范围是一次会话。

**globalsession**：在集群服务器中的整个会话范围。

**scope**：配置作用范围

```
<bean id="userBean" class="com.itheima.spring.UserBean" scope="prototype"></bean>
```

## 5、bean的生命周期

bean的生命周期：

单例的：

出生：一加载spring的配置文件，对象就创建。并且初始化

活着：只要spring容器存在，则该对象一直活着

死亡：spring容器销毁，对象消亡

多例的：

出生：容器创建，对象出生

活着：只要对象被使用，就一直活着

死亡：当对象长时间没有被引用（使用）时，由java的垃圾回收机制，把对象回收。

**init-method**：配置bean对象创建

**destroy-method**：配置了bean对象销毁

```
<bean id="userBean" class="com.itheima.spring.UserBean" scope="prototype" init-method="init"
destroy-method="destroy"></bean>
```

```
public class UserBean {
    //对象出生
    public void init(){
        System.out.println("对象初始化了。。。");
    }
    //对象销毁
    public void destroy(){
        System.out.println("对象销毁了");
    }
}
```

## 6、DI（Dependency Injection 依赖注入）

当我们使用Service对象时，该类中包含了一个Dao对象。

我们希望在获取service对象时，spring就会自动为我们提供dao的对象实例。

简单的说：

依赖注入就是缺什么，传什么。

在spring中使用依赖注入就两种方式：构造函数注入

set方法注入 用的多

**构造函数注入**

涉及的标签：

**constructor-arg**

他就是找对应的构造函数。当有一个该标签，就需要有对应参数列表的构造函数。

属性：

**index**：根据索引位置找。从0开始

**type**：根据数据类型找。

**name**：根据名称找。用的最多就是他

=====

**value**：注入基本数据类型和String类型的值

**ref**：注入的是其他bean类型

```

<bean id="userBean" class="com.itheima.spring.UserBean">
  <constructor-arg name="username" value="test"></constructor-arg>
  <constructor-arg name="age" value="18"></constructor-arg>
  <constructor-arg name="birthday" ref="now"></constructor-arg>
</bean>
<bean id="now" class="java.util.Date"></bean>

```

bean有参构造注入，有几个参数就写几个标签  
参数中的对象类型，需要引用该对象bean

```

public class UserBean {

    private String username = null;
    private Integer age = null;
    private Date birthday = null;

    public UserBean(String username, Integer age, Date birthday) {
        this.username = username;
        this.age = age;
        this.birthday = birthday;
    }
}

```

## set方法注入

涉及的标签：

**property**：使用bean中的set方法给属性赋值

属性：

**name**：找的是bean中的set方法

**value**：注入基本数据类型和String类型的值

**ref**：注入的是其他bean类型

```

<!-- 把资源交给spring来管理
    id:任意写但一个配置文件中不能有相同的ID
    class: 资源文件全路径 -->
<bean id="Jdao" class="com.dao.impls.Jdao"></bean>
<!-- set注入要注意 property name="SDdao" 这个名字的名字要与get注入的get方法名致 否则会报错
    property: 配置资源中的资源归属
    name: 资源的get方法名
    ref: 指向的资源 -->
<bean id="JdbcService" class="com.services.JdbcServices">
  <property name="SDdao" ref="Jdao"></property>
</bean>

```

class指定要 资源类名

注入值：

```

<bean id="userBean2" class="com.itheima.spring.UserBean2">
  <property name="username" value="test1"></property>
  <property name="age" value="11"></property>
  <property name="birthday" ref="now"></property>
</bean>

```

```

public class UserBean2 {

    private String username = null;
    private Integer age = null;
    private Date birthday = null;

    public void setUsername(String username) {
        this.username = username;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
}

```

```

public class JdbcServices implements IjdbcServices {

```

```

    private Ijdao Jdao = null;

    public void setSDdao(Ijdao Jdao) {
        this.Jdao = Jdao;
    }
}

```

## spring注入集合数据

明确：结构相同，标签可以互换

```

<bean id="userBean5" class="com.itheima.spring.UserBean5">
    <!-- 注入数组、list集合、Set集合 可用<array>、<set>、<list> -->
    <property name="myStrs">
        <set>
            <value>AAA</value>
            <value>BBB</value>
            <value>CCC</value>
        </set>
    </property>

```

三个可随意用

```

<!-- 注入Map 注入Properties <props>、<map> -->
<property name="myMap">
    <props>
        <prop key="testCCC">CCC</prop>
        <prop key="testDDD">DDD</prop>
    </props>
</property>

```

类中声明：

```
public class UserBean5 {

    private String[] myStrs;
    private List<String> myList;
    private Set<String> mySet;
    private Map<String, String> myMap;
    private Properties myProps;

    public void setMyStrs(String[] myStrs) {
        this.myStrs = myStrs;
    }

    public void setMyList(List<String> myList) {
        this.myList = myList;
    }
}
```

## 7、spring的多配置文件配置方式

它有两种多配置文件配置的方式：

第一种：并列关系。 bean1.xml,bean1.xml.....

第二种：主从关系

在同一个配置文件中，bean的id不能重复。

在不同的配置文件中，bean的id可以重复，但是后加载的覆盖先加载的。

```
//1. 并列关系, 获取Spring容器
ApplicationContext ac = new ClassPathXmlApplicationContext("bean1.xml", "bean2.xml");
//2. 根据id获取bean对象
UserBean ub1 = (UserBean) ac.getBean("userBean");
```

多个配置文件

主从关系：

新一个新的配置文件引入其它配置文件

```
<!-- 引入其他配置文件 -->
<import resource="bean1.xml"/>
<import resource="bean2.xml"/>
```

## 8、spring整合javaweb工程

spring容器应该是一个应用只有一个。

在应用加载时创建，应用卸载时销毁。

ServletContext对象的生命周期与之相同。

可以通过监听器得知ServletContext对象的创建和销毁。

Spring给我们提供用一个ServletContextListener的实现类，负责监听ServletContext对象的创建和销毁。

### 8.1、我们需要在web.xml配置文件中配置。

配置spring的监听器，用于监听servletContext对象的创建和销毁，保证一个应用只有一个Spring容器

它默认只能加载WEB-INF目录中的名称为applicationContext.xml文件。

```
<listener>
```

```
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

```
</listener>
```

```
<!-- 配置spring的监听器，用于监听servletContext对象的创建和销毁，保证一个应用只有一个Spring容器  
它默认只能加载WEB-INF目录中的名称为applicationContext.xml文件。 -->
```

```
<listener>
```

```
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

```
</listener>
```

### 8.2、将原来放在根目录下的bean.xml改名为applicationContext.xml并将其放在WEB-INF目录

( )

lib	2018/6/24 15:25	文件夹	
applicationContext.xml	2016/12/22 16:16	XML 文件	1 KB
web.xml	2018/7/4 17:37	XML 文件	2 KB

### 8.3、配置之前需要入到一个jar包：spring-web-4.2.4.jar

spring-web-4.2.4.RELEASE.jar	2016/12/22 18:59	Executable Jar File	750 KB
------------------------------	------------------	---------------------	--------

### 8.4、在web项目中获取Spring容器

ApplicationContext ac =

WebApplicationContextUtils.getWebApplicationContext(ServletActionContext.getServletContext());

//2.根据id获取bean对象

customerService = (ICustomerService) ac.getBean("customerService");

```
private ICustomerService customerService;
```

```
public CustomerAction() {
```

```
//1. 在web项目中获取Spring容器
```

```
ApplicationContext ac = WebApplicationContextUtils.getWebApplicationContext(ServletActionContext.getServletContext());
```

```
//2. 根据id获取bean对象
```

```
customerService = (ICustomerService) ac.getBean("customerService");
```

```
}
```



## 9、基于注解的SpringIOC配置

### a、什么是注解：

它就是注释。但是不仅是描述功能用的，而且还是实现功能的重要组成部分。

它不光是给程序员看的，更重要的是给编译器看。

### b、注解的作用

可以实现和配置文件相同的功能，对一些属性进行配置。

### c、XML配置和注解配置选择问题

就看配置是否经常修改。

如果经常修改：用XML

如果不经常修改：可以用注解。因为它的维护方便

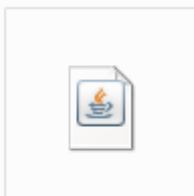
### d、基于注解的IOC配置

搭建开发环境：

第一步：导入必备jar包。

注意：除了ioc的6个之外，还要单独拷贝一个AOP的jar包。

spring-aop-4.2.4.RELEASE.jar



spring-aop-4.2.  
4.RELEASE.jar

注解的IOC配置必备jar

第二步：编写配置文件

在类的根路径下编写一个bean.xml配置文件，并且导入约束

别忘了导入context名称空间和它的约束

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

```
xmlns:context="http://www.springframework.org/schema/context"
```

```
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
```

```
http://www.springframework.org/schema/context
```

<http://www.springframework.org/schema/context/spring-context.xsd>>

<!-- 告知spring在创建容器时要扫描的包

## 1.导入context名称空间

## 2.使用context:compnent-scan配置要扫描的包

-->

```
<context:component-scan base-package="com.itheima">
</context:component-scan>
```

</beans>

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
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
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<!-- 告知spring在创建容器时要扫描的包
1. 导入context名称空间
2. 使用context:compnent-scan配置要扫描的包 非常重要
-->
<context:component-scan base-package="com.itheima"></context:component-scan>
</beans>
```

第三步：使用注解管理资源

@Component把原来的<bean id="" class=""/>用注解替换了

该注解有一个属性：value用于指定bean的id

@Component: 指明当前类是资源类

@Repository: 指明在当前资源类里有一个调用的资源引用对象

@Resource: 指明资源对象对应的资源实体类

```
@Component("customerService")//表明它是一个组件
//就相当于xml的<bean id="customerService" class="com.itheima.service.impl.CustomerServiceImpl"/>
public class CustomerServiceImpl implements ICustomerService {
```

@Component说明当前类是一个资源类

```
@Resource(name="customerDao2")
private ICustomerDao customerDao;
```

@Resource说明当前对象的一个资源引用

```
@Repository("customerDao2")  
public class CustomerDaoImpl2 implements ICustomerDao {
```

第四步：使用context:component-scan来指定创建容器时要扫描的包

```
<context:component-scan base-package="com.itheima"></context:component-scan>
```

e、常用注解：

@Component:

作用：配置让spring来管理资源。相当于

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

属性：

value：指定bean的id。如果不写该属性，默认用类的名称首字母改小写作为bean的id。

@Controller

@Service

@Repository

以上三个注解和@Component的作用是一模一样的。属性也是一模一样的。

他们的出现，只是提供了更加明确的语义化。

上面的四个注解都是用于配置bean对象的。

以下的四个注解都是用于注入的

@Autowired

作用：自动按照类型注入

注意：如果有多个bean的类型都匹配，看变量名称和bean的id有没有一致的。

如果要是有一致的，也可以注入成功。就把一致的那个bean注入。

如果没有一致的，则报错。

@Qualifier

作用：在自动按照类型注入的基础之上，再按照名称注入（bean的id注入）

属性：

value：指定bean的id。

## @Resource

作用：直接按照bean的id注入。

属性：

name: 指定bean的id

以上三个注解都是注入其他bean类型的

## @Value

作用：注入基本数据类型和String类型

属性：

value: 指定要注入的值

## @Scope

作用：配置bean的作用范围的

属性：

value: 指定作用范围。

取值：

singleton: 单例。默认值

prototype: 多例

request

session

globalsession