# 一. Spring框架

## 1 简介

Spring是一种JavaEE应用程序框架，它把JavaEE中的许多功能封装为了单独的工具，以帮助分离项目组件之间的依赖关系，它的主要目地是简化企业开发。

Spring是一个封装了诸多JavaEE功能的工具集。

Spring的核心就是提供一种新的机制管理业务对象及其依赖关系。例如IOC（反转控制），AOP（面向切面编程）

作用：

- 方便解耦，简化开发：Spring 就是一个大工厂，可以将所有对象创建和依赖关系维护，交给 Spring 管理

- AOP 编程的支持：Spring 提供面向切面编程，可以方便的实现对程序进行拦截、运行监控等功能

- 声明式事务的支持：只需要通过配置就可以完成对事务的管理，而无需手动编程

- 方便程序的测试：Spring 对 Junit4 支持，可以通过注解方便的测试 Spring 程序

- 方便集成各种优秀框架：Spring 不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz 等）的直接支持

- 降低 JavaEE API 的使用难度：Spring对 JavaEE 开发中非常难用的API（JDBC、JavaMail、远程调用等），都提供了封装，使这些 API 应用难度大大降低

## 2 核心模块



### core - 核心模块

- spring-core：IoC与DI的最基本实现

- spring-beans：Bean工厂与bean的装配

- spring-context：spring的context上下文即IoC容器

- spring-context-support

- spring-expression：spring表达式语言

详细说明

（1）spring-core

这个jar文件包含Spring框架基本的核心工具类，Spring其它组件要都要使用到这个包里的类，是其它组件的基本核心，当然你也可以在自己的应用系统中使用这些工具类

（2）spring-beans

这个jar文件是所有应用都要用到的，它包含访问配置文件、创建和管理bean以及进行Inversion of Control / Dependency Injection（IoC/DI）操作相关的所有类。如果应用只需基本的IoC/DI支持，引入spring-core.jar及spring- beans.jar文件就可以了

（3）spring-context

Spring核心提供了大量扩展，这样使得由 Core 和 Beans 提供的基础功能增强：这意味着Spring 工程能以框架模式访问对象。Context 模块继承了Beans 模块的特性并增加了对国际化（例如资源绑定）、事件传播、资源加载和context 透明化（例如 Servlet container）。同时，也支持JAVA EE 特性，例如 EJB、 JMX 和 基本的远程访问。Context 模块的关键是 ApplicationContext 接口。spring-context-support 则提供了对第三方库集成到 Spring-context 的支持，比如缓存（EhCache, Guava, JCache）、邮件（JavaMail）、调度（CommonJ, Quartz）、模板引擎（FreeMarker, JasperReports, Velocity）等。

（4）spring-expression

为在运行时查询和操作对象图提供了强大的表达式语言。它是JSP2.1规范中定义的统一表达式语言的扩展，支持 set 和 get 属性值、属性赋值、方法调用、访问数组集合及索引的内容、逻辑算术运算、命名变量、通过名字从Spring IoC容器检索对象，还支持列表的投影、选择以及聚合等。

### 数据访问与集成层

包含 JDBC、ORM、OXM、JMS和事务模块。

（1）spring-jdbc

提供了 JDBC抽象层，它消除了冗长的 JDBC 编码和对数据库供应商特定错误代码的解析。

（2）spring-tx

支持编程式事务和声明式事务，可用于实现了特定接口的类和所有的 POJO 对象。编程式事务需要自己写beginTransaction()、commit()、rollback()等事务管理方法，声明式事务是通过注解或配置由 spring 自动处理，编程式事务粒度更细。

（3）spring-orm

提供了对流行的对象关系映射 API的集成，包括 JPA、JDO 和 Hibernate 等。通过此模块可以让这些 ORM 框架和 spring 的其它功能整合，比如前面提及的事务管理。

（4）spring-oxm

模块提供了对 OXM 实现的支持，比如JAXB、Castor、XML Beans、JiBX、XStream等。

（5）spring-jms

Spring 处理Web层

Web 层包括 spring-web、spring-webmvc、spring-websocket、spring-webmvc-portlet 等模块。

详细说明

（1）spring-web

提供面向 web 的基本功能和面向 web 的应用上下文，比如 multipart 文件上传功能、使用 Servlet 监听器初始化 IoC 容器等。它还包括 HTTP 客户端以及 Spring 远程调用中与 web 相关的部分

（2）spring-webmvc

为 web 应用提供了模型视图控制（MVC）和 REST Web 服务的实现。Spring 的 MVC 框架可以使领域模型代码和 web 表单完全地分离，且可以与 Spring 框架的其它所有功能进行集成

（3）spring-webmvc-portlet

（即Web-Portlet模块）提供了用于 Portlet 环境的 MVC 实现，并反映了 pring-webmvc 模块的功能

Portal是一个基于Web的应用，它能提供个性化，单点登录，不同源的内容聚合，和信息系统的表示层集中

### Spring AOP

（1）spring-aop

提供了面向切面编程（AOP）的实现，可以定义诸如方法拦截器和切入点等，从而使实现功能的代码彻底的解耦。使用源码级的元数据。

（2）spring-aspects

提供了对 AspectJ 的集成

### Instrumentation 模块

（1）spring-instrument

模块提供了对检测类的支持和用于特定的应用服务器的类加载器的实现。

（2）spring-instrument-tomcat

模块包含了用于 Tomcat 的Spring 检测代理。

### Messaging消息处理

spring-messaging 模块

从 Spring 4 开始集成，从一些 Spring 集成项目的关键抽象中提取出来的。这些项目包括 Message、MessageChannel、MessageHandler 和其它服务于消息处理的项目。这个模块也包含一系列的注解用于映射消息到方法

### Test模块

spring-test 模块

通过 JUnit 和 TestNG 组件支持单元测试和集成测试。它提供了一致性地加载和缓存 Spring 上下文，也提供了用于单独测试代码的模拟对象（mock object）

# 二 开发步骤

## 1 导入jar包

本教程使用spring5.1.5

Maven方式：

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.1.5.RELEASE</version>

</dependency>

## 2 核心配置文件

习惯上名字为

applicationContext.xml/bean.xml/spring-XXXX.xml

使用的约束可以在帮助文档中找到

<?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"*>

<!-- IOC容器的配置 通过bean配置 创建对象 -->

<bean id=*"user"* class=*"com.rr.base.User"*></bean>

</beans>

bean的属性：

name属性

不可以重复，可以使用特殊字符

id属性

唯一的，不可以重复，id属性作用和name几乎相同

scope属性

设置bean对应对象生成规则.

scope = "singleton" 单例,默认值,适用于实际开发中的绝大部分情况.

scope="prototype"多例,适用于struts2中的action配置

lazy-init属性

只对单例有效,设置scope="singleton"时测试

lazy-init="false" 默认值,不延迟创建,即在启动时候就创建对象.

lazy-init="true" 延迟初始化,在用到对象的时候才会创建对象.

## 3 调用

//获取ApplicationContext对象

ApplicationContext ac = **new** ClassPathXmlApplicationContext("com/rr/base/applicationContext.xml");

//从IOC容器中获取配置的对象

User u = (User)ac.getBean("user");

System.***out***.println(u);

# 三 spring容器

## 1 IOC控制反转

IOC： Inversion Of Control

所谓控制反转就是应用本身不负责依赖对象的创建及维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是所谓反转，目的是为了获得更好的扩展性和良好的可维护性。

通过IOC，开发人员不需要关心对象的创建过程，交给Spring容器完成，可以降低对象之间的耦合关系。具体的过程是，程序读取Spring 配置文件，获取需要创建的 bean 对象，通过反射机制创建对象的实例。

1)调用无参数构造器

<bean id="p1" class="com.rr.ioc.Person"></bean>

2)带参数构造器

<bean id=*"p2"* class=*"com.rr.ioc.Person"*>

<!-- 参数赋值顺序要和构造方法中参数顺序一致 -->

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

<constructor-arg value=*"20"*></constructor-arg>

</bean>

<bean id=*"p3"* class=*"com.rr.ioc.Person"*>

<!-- 通过index指定参数的索引 -->

<constructor-arg index=*"1"* value=*"20"*></constructor-arg>

<constructor-arg index=*"0"* value=*"zhangsan"*></constructor-arg>

</bean>

<!-- String str = new String("hello"); -->

<bean id=*"str"* class=*"java.lang.String"*>

<constructor-arg value=*"hello"*></constructor-arg>

</bean>

<bean id=*"p4"* class=*"com.rr.ioc.Person"*>

<!-- 通过ref引用其他的bean ref的值，是其他bean的id值 -->

<constructor-arg index=*"1"* value=*"20"*></constructor-arg>

<constructor-arg index=*"0"* ref=*"str"*></constructor-arg>

</bean>



3)静态方法创建对象

<bean id=*"p5"* class=*"com.rr.ioc.MethodTest"* factory-method=*"getInstance"*></bean>

## 2 DI依赖注入

DI：Dependency Injection

所谓依赖注入就是指：在运行期，由外部容器动态地将依赖对象注入到另一个对象中。

A a = new A()

B b = new B();

a.setB(b)

1)通过构造函数

2)通过set方法给属性注入值



<bean id=*"d1"* class=*"com.rr.di.Dog"*>

<property name=*"name"* value=*"wangcai"*></property>

</bean>

<!-- dao -->

<bean id=*"dogDao"* class=*"com.rr.di.DogDao"*></bean>

<!-- service -->

<bean id=*"dogService"* class=*"com.rr.di.DogService"*>

<property name=*"dd"* ref=*"dogDao"*></property>

</bean>

<!-- action -->

<bean id=*"dogAction"* class=*"com.rr.di.DogAction"*>

<property name=*"ds"* ref=*"dogService"*></property>

</bean>

3）p标签注入

增加如下约束：

xmlns:p=*"http://www.springframework.org/schema/p"*

<bean id=*"name"* class=*"java.lang.String"*>

<constructor-arg value=*"wangwu"*></constructor-arg>

</bean>

<!-- p:属性名 -->

<bean id=*"dog"* class=*"com.qianfeng.di.Dog"* p:age=*"10"* p:name=*"wangcai"*>

</bean>

<!-- p:属性名-ref， 引入其他对象 -->

<bean id=*"dog2"* class=*"com.qianfeng.di.Dog"* p:age=*"10"* p:name-ref=*"name"*>

</bean>

4)自动注入autowire

byType 根据类型注入

针对相同的类型，如果有多个对象，根据类型注入时会报错

byName 根据名称注入

## 3注解

@Component 所有受Spring管理组件的通用形式

@Controller标识控制层组件

@Service标识业务层组件

@Repository标识持久层组件

@Resource用注解组装bean（注入），java提供的注解，默认根据名称进行注入

如果通过@Resource(name=”xxx”),只会根据名字进行注入；

如果没有指定，会根据修饰的变量的默认名注入，如果不存在，根据类型注入，如果有多个相同接口的类，报错，可以通过type属性指定具体的类

@Resource(type=UserDaoImpl2.**class**)

@Autowried 根据类型自动注入对象，如果匹配多个类型一致的对象（比如一个接口多个实现类），需要使用@Qualifier()根据名称选择具体注入哪一个对象

@Scope 设置bean的作用域，@Scope(scopeName="singleton")

注：需要导入aop相关的jar文件

spring-aop-5.1.5.RELEASE.jar

引入相关约束（绿色所示配置）：

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

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

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

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"*>

<!-- 注解扫描 自动扫描本包和它的子包下的注解 -->

<context:component-scan base-package="com.rr.anno"></context:component-scan>