# Spring 基础

## Spring 简介

### Spring 是什么

1. Spring 是一个工厂（Factory）
2. Spring是一个容器（Container）
3. Spring 是一个框架（Framework）

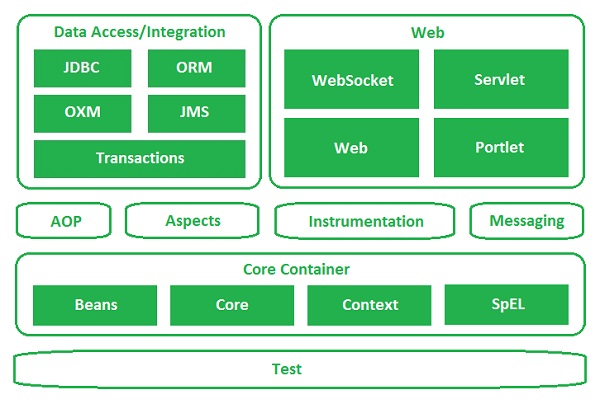
### Spring 应用动机

1. 优化对象创建、管理，降低侵入性。
2. 简化应用开发，降低耦合。
3. 解耦并提供整体解决方案

实际企业项目开发中我们通常会借助Spring整合hibernate,mybatis等技术框架实现快速高效开发。

### Spring 基本架构

Spring 框架的基本结构图如下：



Spring框架中最核心的为IOC（控制反转），并基于IOC实现了AOP，整合了数据访问及Web组件功能。

# Spring IOC 容器基本应用

## Spring IOC 概述

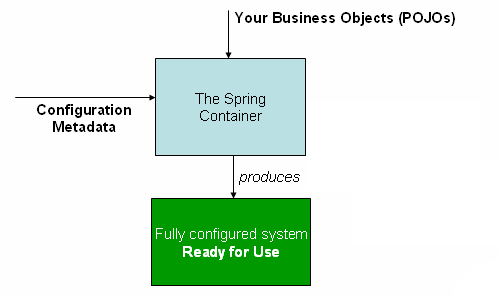
IoC全称是Inversion of Control，被译为控制反转，是指程序中对象的获取方式发生反转，由最初的new方式创建，转变为由第三方框架创建、注入（DI），它降低了对象之间的耦合度。

Spring容器是IOC机制的一种实现，同时IOC也是Spring框架的的基础和核心，它借助DI（Dependency Injection）方式实现。

## Spring Bean容器

Spring容器是Spring框架中的核心组件，负责创建Bean对象(一种简单规范的java对象)及管理这些对象之间的依赖关系，。

Spring容器是如何工作的呢？在Spring项目运行时， Spring容器负责读取项目中的元数据信息（这些元数据信息可能是基于xml也可能是基于注解实现），然后基于这些信息创建bean对象实例，如下图所示：



IOC 是Spring框架中的一种机制，提供了容器中对象的控制反转功能，这个功能的实现需要借助依赖查找和依赖注入。

### Spring 容器元数据配置

Spring中元数据的配置有两种方式，基于xml或annotation方式。这个小节了解

基于xml方式的元数据实现bean元素的基本配置。

<?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">

<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>

Spring中多个元数据配置文件的实现方式

<beans>

<import resource="services.xml"/>

<import resource="resources/messageSource.xml"/>

<import resource="/resources/themeSource.xml"/>

<bean id="bean1" class="..."/>

<bean id="bean2" class="..."/>

</beans>

### Spring 容器初始化

Spring中容器的类型为Application类型，其初始化方式如下：

ApplicationContext context =

**new** ClassPathXmlApplicationContext("beans.xml");

Spring容器初始化时需要加载元数据的配置信息，例如beans.xml文件中声明的元数据的相关配置。

### Spring 容器的使用

Spring容器初始化以后，可以通过容器的getBean(…)方法获取容器中的Bean对象，进而操作Bean对象，例如

Contex.getBean(“xxxService”,XxxService.class);

## Spring Bean对象

### Spring bean 对象的命名

Spring容器中，每个Bean都需要有自己的名字，这个名字的定义需要符合标识符的规范，然后借助id或name属性指定。

### Spring对象的实例化

Spring容器创建Bean对象的方法有如下3中方式：

1. 通过构造器实例化Bean对象
2. 通过静态工厂方法实例化Bean对象
3. 通过实例工厂方法实例化Bean对象。

<bean id="date1" class="java.util.Date"/>

<bean id="cal" class="java.util.Calendar"

factory-method="getInstance"/>

<bean id="date2" factory-bean="cal"

factory-method="getTime"/>

### Spring bean 对象的作用域

在理解Bean的作用域时重点掌握如下两个作用域：

1. Singleton spring容器中此bean的实例只有一份
2. Prototype spring 容器中每次获取bean都会创建新一个新的对象。

其作用域的配置可记住bean标签的scope属性进行指定，例如

<bean id="helloService"

class="com.company.spring.factory.HelloServiceImpl"

scope="singleton"/>

### Spring bean 对象的生命周期

Spring bean对象的生命周期由Spring容器进行管理，并可以在bean元素定义时借助init-method属性指定初始化时调用的方法，借助detroy-method属性指定销毁时要调用的方法（销毁回调方法只适用于singleton作用域范围的对象）。

<bean id="helloService"

class="com.company.spring.factory.HelloServiceImpl"

scope="singleton"

init-method="doInit"

destroy-method="doDestory"/>

### Spring bean 对象的延迟加载

Spring 容器默认会在容器启动时将所有singleton 作用域下的bean进行实例化，如果不想启动时实例化，而是在第一次使用时实例化，可以通过bean元素定义时的lazy-init=“true”的方式进行指定。

<bean id=*"helloService"*

class=*"com.company.spring.factory.HelloServiceImpl"*

scope=*"singleton"*

init-method=*"doInit"*

destroy-method=*"doDestory"*

lazy-init=*"true"*/>

提示：假如希望容器中所有的Bean都要采用延迟加载，则可以在配置文件的顶级<beans/>元素中设置default-lazy-init属性的值为true.

## Spring Bean依赖

### 依赖注入基础

Spring 容器中的Bean对象通常会存在一定的依赖关系，而这种依赖关系的实现在Spring 框架中要借助于DI机制。其中DI就是借助对象管理对象依赖关系的一个过程。

Spring中提供的依赖注入方式有构造注入和设置注入，其中构造注入就是借助构造方法的参数实现对类中属性值的注入，set注入就是借助set方法的参数实现其属性值的注入。

Spring 依赖注入时，可以实现基本值的注入，Bean对象的注入，集合的注入，spring表达式方式的注入等等。

set注入：（重点掌握）

<bean id="dataSource1"

class="com.company.spring.util.DataSource">

<property name="driverClassName"

value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql:///test"/>

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

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

</bean>

<bean id="jdbcTemplate1"

class="com.company.spring.injection.JdbcTemplate">

<property name="dataSource" ref="dataSource1"/>

</bean>

constructor注入：(了解)

<bean id="dataSource2" class="com.company.spring.util.DataSource">

<constructor-arg value="com.mysql.jdbc.Driver"/>

<constructor-arg value="jdbc:mysql:///test"/>

<constructor-arg value="root"/>

<constructor-arg value="root"/>

</bean>

<bean id="jdbcTemplate2"

class="com.company.spring.injection.JdbcTemplate">

<constructor-arg ref="dataSource2"/>

</bean>

### 依赖注入高级

实际项目中对象的属性可能会有数组类型，集合等类型，这些类型的注入的方式可参考如下案例实现。

定义一个相对复杂的对象

**public** **class** ComplexObject {

**private** String[] hobby;

**private** List<String> address;

**private** Map<String,String> phones;

**private** Properties configs;

//set,get,…

}

对象在配置文件中的实现：

<bean id="complexObject"

class="com.company.spring.injection.ComplexObject">

<property name="hobby">

<list>

<value>足球</value>

<value>篮球</value>

</list>

</property>

<property name="address">

<list>

<value>北京</value>

<value>上海</value>

</list>

</property>

<property name="phones">

<map>

<entry key="pKey1" value="123"/>

<entry key="pKey2" value="456"/>

</map>

</property>

<property name="configs">

<props>

<prop key="cKey1">#{cfg.port}</prop>

<prop key="cKey2">#{cfg.host}</prop>

</props>

</property>

</bean>

<util:properties id="cfg"

location="classpath:config.properties"/>

提示：

其中util:properties元素用于引入配置文件，并可以通过spring表达式获取对应key的值，然后注入给其它对象的属性。

### 自动装配实现

Spring IoC容器可以自动装配（autowire）相互协作bean之间的关联关系，autowire可以针对单个bean进行设置，autowire的方便之处在于减少xml的注入配置

Spring 配置文件中通过bean元素的autowire属性指定自动装配规则，一共有四种类型值

|  |  |  |
| --- | --- | --- |
|  | 属性值 | 描述 |
|  | NO | 禁用自动配置（默认） |
|  | ByName | 按名字自动装配(重点掌握) |
|  | ByType | 按类型自动状态（重点掌握），但有多个类型时会出错 |
|  | Constructor | 与byType类似，不同之处在于它应用于构造器参数。 |

例如：

<bean id="jdbcTemplate3"

class="com.company.spring.injection.JdbcTemplate"

autowire="constructor">

</bean>

<bean id="jdbcTemplate3"

class="com.company.spring.injection.JdbcTemplate"

autowire="byName">

</bean>

# Spring IOC 容器注解应用

## Bean组件扫描

### 组件扫描定义

Spring中通过指定一个包路径，由系统自动扫描该包及其子包所有组件类,当发现组件类定义前有特定的注解标记时，就将该组件纳入到Spring容器。

这种方式的配置等价于原有XML配置中的<bean>定义功能组件扫描，可以替代大量XML配置中的<bean>元素的定义

### 扫描方式配置

使用组件扫描，首先需要在XML配置中指定扫描父级package路径，例如

<context:component-scan base-package=”com.company.spring/>”

在这个配置中，容器会自动扫描org.example包及其子包下所有组件，并实例化bean对象。

## 系统组件注解

### 组件注解标记

指定扫描类路径后，并不是该路径下所有组件类对象都由Spring容器创建并管理，只有在组件类定义前面有以下注解标记时，才会扫描到Spring容器。

|  |  |  |
| --- | --- | --- |
|  | 注解名 | 说明 |
|  | @Component | 通用注解 |
|  | @Repository | 持久层组件应用注解 |
|  | @Service | 业务层组件应用注解 |
|  | @Controller | 控制层组件应用注解 |

提示：实际项目中某个业务的实现，是要进行分层处理的，每一层的对象都应该有自己的职责，目的是便于维护与扩展。

例如：

数据层对象

@Repository

**public** **class** SysUserDaoImpl **implements** SysUserDao{

**public** **void** saveUser(Object obj) {

System.***out***.println("dao.save.user");

}

}

业务层对象

@Service

**public** **class** SysUserServiceImpl **implements** SysUserService{

…

}

控制层对象

@Controller

**public** **class** UserController {

}

### 系统组件命名

当一个组件在扫描过程中被检测到时，会生成一个默认id值，默认id为小写开头的类名。也可以在注解标记中自定义id，例如

@Component

**public** **class** IdGenerator {

**public** IdGenerator() {

System.***out***.println("IdGenerator");

}

}

### 系统组件作用域

通常受Spring管理的组件，默认的作用域是"singleton"。如果需要其他的作用域可以使用@Scope注解，只要在注解中提供作用域的名称即可。

@Scope("singleton")

@Component

**public** **class** IdGenerator {

**public** IdGenerator() {

System.***out***.println("IdGenerator");

}

}

在Bean组件中还可以借助@PostConstruct和@PreDestroy注解标记指定

初始化和销毁回调方法。这个知识点了解即可。

@Scope("singleton")

@Component

**public** **class** IdGenerator {

**public** IdGenerator() {

System.***out***.println("IdGenerator");

}

@PostConstruct

**public** **void** init() {

System.***out***.println("init()");

}

@PreDestroy

**public** **void** destory() {

System.***out***.println("destory()");

}

}

### 组件依赖关系注解

实际项目中对象与对象之间通常会具有一定的依赖关系，这种关系假如基于spring注解的方式进行注入，可使用@Autowired/@Qualifier,@Resource注解在对象的属性或set方法上进行声明。

* @Autowired注解应用

@Service

**public** **class** SysUserServiceImpl **implements** SysUserService{

@Autowired

**private** SysUserDao userDao;

**public** **void** saveUser(Object obj) {

userDao.saveUser(obj);

}

}

其中@Autowire注解默认按类型进行注入，假如希望按名字进行注入，可以再结合@Qualifier一起使用，例如

@Service

**public** **class** SysUserServiceImpl **implements** SysUserService{

@Autowired

@Qualifier("userDaoImpl")

**private** SysUserDao userDao;

**public** **void** saveUser(Object obj) {

userDao.saveUser(obj);

}

}

* @Resource注解应用

@Controller

**public** **class** SysUserController {

@Resource

**private** SysUserService userService;

**public** **void** doSaveUser(Object obj) {

userService.saveUser(obj);

}

}

其中@Resource属于JavaEE中的一个注解，默认按对应的属性名进行装配注入，假如希望指定的名称进行注入，可通过注解中的name属性进行配置，例如

@Controller

**public** **class** SysUserController {

@Resource(name="userServiceImpl")

**private** SysUserService userService;

**public** **void** doSaveUser(Object obj) {

userService.saveUser(obj);

}

}

### 组件注入表达式应用

Spring应用中可借助@Value注解通过Spring表达式获取对应properties文件中的值，并将其值注入到对应对象的属性上。

首先在XML配置中指定要注入的properties文件，例如

config.properties文件内容

prefix=CGB

suffix=JT

spring中配置文件中添加如下语句。

<util:properties

id=*"cfg"* location=*"classpath:config.properties"*/>

然后在属性或Setter方法前使用@Value注解

@Scope("singleton")

@Component

**public** **class** IdGenerator {

@Value("#{cfg.prefix}")

**private** String prefix;

@Value("#{cfg.suffix}")

**private** String suffix;

…

}

# Spring IOC 总结

## 重点难点

## FAQ