极客学院
jikexueyuan.com

简化Spring XML的配置

简化Spring XML的配置 — **课程概要**

- 自动装配Bean的属性
- 基于注解的配置
- 基于Java类的配置
- 不同配置方式的比较

自动装配Bean的属性

自动装配Bean的属性

当Spring装配Bean的属性时，如果非常明确，则可以使用自动装配模式：

- 自动装配类型
- 默认自动装配
- 混合装配

自动装配Bean的属性 – **自动装配类型**

Spring提供了4种各具特色的自动装配策略：

- byName
- byType
- constructor
- autodetect

自动装配Bean的属性 – **自动装配类型**

默认情况下，不自动装配，通过“ref”标签手动设定：

类文件：

```
public class Customer
{
   private  Person  person;
   public void setPerson(Person person) {
      this.person = person;
   }
}

public class Person {……}
```

配置文件：

```
<bean id="customer" class="com.jike.***.Customer">
        <property name="person" ref="person" />
</bean>
 <bean id="person" class="com.jike.***.Person" />
```

自动装配Bean的属性 – **自动装配类型**

byName自动装配：

```
<bean  id="customer" class="com.jike.***.Customer”  autowire="byName" />
<bean  id="person" class="com.jike.***.Person" />
```

byType自动装配：

```
<bean id="customer" class="com.jike.***.Customer”  autowire="byType" />
<bean id="person" class="com.jike.***.Person"  />
```

```
<bean id="customer" class="com.jike.***.Customer”  autowire="byType" />
<bean id="person1" class="com.jike.***.Person"  primary=“false”  />
```

```
<bean id="customer" class="com.jike.***.Customer”  autowire="byType" />
<bean id="person2" class="com.jike.***.Person"  autowire-candidate=“false”  />
```

自动装配Bean的属性 – **自动装配类型**

constructor自动装配：

```
<bean id="customer" class="com.jike.***.Customer”  autowire=“constructor" />
<bean id="person" class="com.jike.***.Person"  />
```

autodetect自动装配：

```
<bean id="customer" class="com.jike.***.Customer”  autowire=“autodetect" />
<bean id="person" class="com.jike.***.Person"  />
```

自动装配Bean的属性 – **默认自动装配**

当Spring要为它所创建的所有Bean应用相同的自动装配策略来简化配置时，可以在根元素<bean>上增加一个default-autowire属性：

```
<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-3.0.xsd”
          default-autowire=“byType”  >
          ……
</beans>
```

自动装配Bean的属性 – **混合装配**

当我们对某个Bean选择了自动装配策略时，仍然可以为任意一个属性配置<property>属性，即可以同时使用自动装配和显式装配策略：

```
<bean id="customer" class="com.jike.***.Customer“autowire=“byType" >
    <property name="person" ref="person1" />
</bean>
 <bean id="person"     class="com.jike.***.Person" />
 <bean id="person1"   class="com.jike.***.Person" />

<bean id="customer" class="com.jike.***.Customer“autowire=“byType" >
    <property name="person" ><null/></property>
</bean>
 <bean id="person"    class="com.jike.***.Person" />
 <bean id="person1"  class="com.jike.***.Person" />
```

基于注解的配置

基于注解的配置

Spring2.0开始引入基于注解的配置方式，即Bean的定义信息可以通过在Bean的实现类上标注注解实现：

- 注解配置示例
- 加载注解配置
- 常用注解详解

基于注解的配置 – 注解配置示例

@Component 是Spring容器中的基本注解，表示容器中的一个组件（bean），可以作用在任何层次，下面的示例介绍该注解的使用方法：

注解配置示例：

```
@Component(“userDao”)
public  class   UserDao { …… }
```

等效XML配置：

```
<bean id=“userDao”  class=“com.jike.***.UserDao” />
```

可用作定义Bean的注解：

- @Component
- @Controller
- @Repository
- @Service

基于注解的配置 – **加载注解配置**

Spring在2.5后提供了一个context的命名空间，它提供了通过扫描类包来加载利用注解定义的Bean的方式：

```
<?xml version="1.0" encoding="UTF-8"?>
<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"   ①
        xsi:schemaLocation="......
            http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <context:component-scan  base-package="com.jike.spring"/>  ②
</beans>
```

基于注解的配置 – 加载注解配置

过滤方式之resource-pattern

```
<context:component-scan     base-package="com.jike.spring”
                            resoure-pattern=“anno/*.class” />
```

过滤方式之过滤子元素

```
<context:component-scan     base-package="com.jike.spring >
  <context:include-filter type=“regex” expression=“com.jike.spring.*” />
  <context:exclude-filter type=“aspectj” expression=“com.jike..*Controller+” />
</context:component-scan>
```

基于注解的配置 – 加载注解配置

过滤表达式

类别	示例	说明
annotation	com.jike.XxxAnnotation	符合XxxAnnotation的target class
assignable	com.jike.XxxService	指定class或interface的全名
aspectj	com.jike…*Service+	AspectJ语法
regex	Com\.jike\.Default\.*	Regelar Expression
Custom	com.jike.MyTypeFilter	Spring3新增自定Type，实作 org.springframework.core.type.TypeFilter

基于注解的配置 – **常用注解详解**

Spring 3.0 提供了一系列的针对依赖注入的注解，这使得 Spring IoC 在 XML 文件之外多了一种可行的选择，主要包含如下注解类型：

- Bean的定义注解
- Bean的生命周期注解
- Bean的依赖检查注解
- Bean的自动装配注解

基于注解的配置 – 常用注解详解（定义注解）

Spring 自 2.0 开始，陆续引入了一些注解用于简化 Spring 的开发。@Repository 注解便属于最先引入的一批，用于将数据访问层 (DAO 层) 的类标识为 Spring Bean：

① 使用 @Repository 将 DAO 类声明为 Bean

```
@Repository
public class UserDaoImpl  implements  UserDao{ …… }
```

② 在 XML 配置文件中启动 Spring 的自动扫描功能

```
<beans … >
    <context:component-scan base-package=“com.jike.dao” />
    ……
</beans>
```

基于注解的配置 – 常用注解详解（Bean定义注解）

Spring 2.5 在 @Repository 的基础上增加了功能类似的额外三个注解，共有如下四种注解：

- @Component：一个泛化的概念，表示一个组件 (Bean)，可作用在任何层次；
- @Repository：用于对DAO实现类进行标注；
- @Service： 用于对Service实现类进行标注；
- @Controller：用于对Controller实现类进行标注；

```
<beans ...>
    <context:component-scan  base-package= “com.jike”
      name-generator= “com.jike.SimpleNameGenerator"/>
</beans>
```

```
@Scope("prototype")
@Repository
public class Demo { … }
```

基于注解的配置 – 常用注解详解（生命周期注解）

在某些情况下，可能需要我们手工做一些额外的初始化或者销毁操作，例如资源的获取和释放操作，Spring 1.x 为此提供了两种方式供用户指定执行生命周期回调的方法：

- 实现 Spring 提供的两个接口：InitializingBean 和 DisposableBean
- 在 XML 文件中使用 <bean> 的 init-method 和 destroy-method 属性

```
<bean id="userService"      class="com.jike.***.UserService"
          init-method="init"  destroy-method="destroy" >
</bean>
```

基于注解的配置 – 常用注解详解（生命周期注解）

Spring 2.5 在保留以上两种方式的基础上，提供了对 JSR-250 的支持。JSR-250 规范定义了两个用于指定声明周期方法的注解：

- @PostConstruct：初始化之后执行的回调方法
- @PreDestroy：销毁之前执行的回调方法

注解示例说明：

```
public class PersonService {
    @PostConstruct
    public void  init(){ ……}
    @PreDestroy
    public void  dostory(){ …… }
}
```

配置文件示例说明：

```
<context:annotation-config />
```

基于注解的配置 – 常用注解详解（依赖检查注解）

Spring 2.0之前使用 dependency-check 在配置文件中设置，属性用于进行依赖检查，缺点是粒度较粗；该属性的取值包括以下几种：

- none -- 默认不执行依赖检查
- simple -- 对原始基本类型和集合类型进行检查
- objects -- 对复杂类型进行检查
- all -- 对所有类型进行检查

使用 Spring2.0 提供的 @Required 注解，提供了更细粒度的控制，@Required 注解只能标注在 Setter 方法之上：

```
<context:annotation-config />
```

基于注解的配置 – 常用注解详解（自动装配注解）

@Autowired可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作，它根据类型进行自动装配，如需按名称进行装配，则需要配合@Qualifier使用：

@Autowired示例：

```
@Service①
public class LogonService {
    @Autowired②
    private LogDao logDao;
}
```

required示例：

```
public class LogonService {
    @Autowired（required=false）③
    private LogDao logDao;
}
```

@Qualifier示例：

```
public class LogonService {
    @Autowired
    @Qualifier(“userDao”)④
    private UserDao userDao;
}
```

基于注解的配置 – 常用注解详解（自动装配注解）

@Autowired可以对类成员变量以及方法的入参进行标注，如下所示：

@Autowired标注方法入参示例：

```
public class LogonService {
   @Autowired①
    public void setLogDao(LogDao logDao) {
          this.logDao = logDao;
    }
    @Autowired
    @Qualifier(“userDao”) ②
    public void setUserDao(UserDao userDao) {
          this.userDao = userDao;
    }
}
```

@Qualifier用于方法入参示例：

```
@Autowired
public void init(@Qualifier(“userDao”)UserDao userDao) {③
        this.userDao = userDao;
}
```

基于注解的配置 – 常用注解详解（自动装配注解）

@Autowired可以对类中集合类的变量或方法入参进行标注，此时会将容器中类型匹配的所有Bean都自动注入进来，如下所示：

@Autowired标注集合入参示例：

```
public class LogonService {
    @Autowired(required=false) ①
    public List<Plugin> plugins ;
    public List<Plugin> getPlugins() {
    return plugins;
    }
}
```

基于Java类的配置

基于Java类的配置

基于Java类定义Bean配置元数据，其实就是通过Java类定义Spring配置元数据，且直接消除XML配置文件：

- 基于Java类的配置示例
- @Configuration注解介绍
- @Bean注解介绍
- 结合基于Java和基于XML方式的配置
- 使用基于Java类的配置信息启动Spring容器

基于注解的配置 – 基于Java类的配置示例

首先让我们看一下基于Java类如何定义Bean配置元数据，具体步骤如下：

- @Configuration注解需要作为配置的类
- @Bean注解相应的方法
- AnnotationConfigApplicationContext或子类进行加载

配置类示例：

```
@Configuration
public class ApplicationContextConfig {
   @Bean
   public String message() {
      return "hello";
   }
}
```

加载类示例：

```
public class ConfigurationTest {
   public static void main(String[] args) {
      AnnotationConfigApplicationContext ctx =
      new AnnotationConfigApplicationContext(ApplicationContextConfig.class);
      System.out.println(ctx.getBean("message"));
   }
}
```

基于注解的配置 – @Configuration注解介绍

通过@Configuration注解的类将被作为配置类使用，表示在该类中将定义Bean配置元数据，且使用@Configuration注解的类本身也是一个Bean，使用方式如下所示：

@Configuration注解示例：

```
@Configuration("ctxConfig")
public class ApplicationContextConfig {
    ......
}
```

基于注解的配置 – @Bean注解介绍

过@Bean注解配置类中的相应方法，则该方法名默认就是Bean名，该方法返回值就是Bean对象，并定义了Spring IoC容器如何实例化、自动装配、初始化Bean逻辑，具体使用方法如下：

@Bean注解格式：

```
@Bean(name={},
    autowire=Autowire.NO,
    initMethod="",
    destroyMethod="")
```

@Bean注解示例：

```
@Bean
public String message() {
    return new String("hello");
}
```

等价XML配置：

```
<bean id="message" class="java.lang.String">
    <constructor-arg index="0" value="hello"/>
</bean>
```

基于注解的配置 – 结合基于Java和基于XML的配置

基于Java方式的配置方式不是为了完全替代基于XML方式的配置，两者可以结合使用，因此可以有两种结合使用方式：

- 在基于Java方式的配置类中引入基于XML方式的配置文件
- 在基于XML方式的配置文件中中引入基于Java方式的配置

引入基于XML配置文件：

```
<bean id="message" class="java.lang.String">
  <constructor-arg index="0" value="test"></constructor-arg>
</bean>

@Configuration("ctxConfig")
@ImportResource("classpath:com/jike/***/appCtx.xml")
public class ApplicationContextConfig {
        ……
}
```

引入基于Java的配置文件：

```
<context:annotation-config/>
<bean id="ctxConfig" class=“com.jike.***..ApplicationContextConfig"/>


public void testXmlConfig() {
    String configLocations[] = {" classpath:com/jike/***/appCtx.xml"};
    ApplicationContext ctx = new ClassPathXmlApplicationContext(configLocations);
     ……
}
```

基于注解的配置 – 启动Spring容器

Spring提供了一个AnnotationConfigApplicanContext类，能够直接通过标注@Configuration的Java类启动Spring容器：

通过构造函数加载配置类:

```
ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConf.class);
```

通过编码方式注册配置类：

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
ctx.register(DaoConfig.class);
ctx.register(ServiceConfig.class);
ctx.refresh();
```

引入多个配置类：

```
@Configuration
@Import(DaoConfig.class)
public class ServiceConfig {……}
```

不同配置方式比较

不同配置方式比较

	基于XML配置	基于注解配置	基于Java类配置
Bean定义	在XML文件中通过<bean>元素定义Bean	在Bean实现类处通过标注@Component等定义Bean	在标注了@Configuration的Java类中，在类方法上标注@Bean定义Bean
Bean名称	通过<bean>的id或name属性定义	通过注解的value属性定义，如@Component（”userDao”）	通过@Bean的name属性定义，如@Bean（”userDao”）
Bean注入	通过<property>子元素或通过p命名空间的动态属性注入	通过标出@Autowired，按类型匹配自动注入，课配合使用@qualifier按名称匹配注入	1.方法处通过@Autowired是方法入参绑定Bean 2.通过调用配置类的@Bean方法进行注入
Bean生命过程方法	通过<bean>的init-method和destroy-method属性指定Bean实现类方法名。	通过在目标方法上标注@PostConstruct和@PreDestroy注解指定	通过@Bean的initMethod或destoryMethod指定相应方法
Bean作用范围	通过<bean>的scope属性指定	通过在类定义出标注@Scope指定	通过Bean方法定义出标注@Scope指定
Bean延迟初始化	通过<bean>的lazy-init属性指定，默认为default	通过在类定义出标注@Lazy指定，如@Lazy（true）	通过在Bean方法定义出标注@Lazy指定

不同配置方式比较

基于XML的配置：

- 第三方类库，如DataSource、JdbcTemplate等；
- 命名空间，如aop、context等；

基于注解的配置：

- Bean的实现类是当前项目开发的，可直接在Java类中使用注解配置

基于Java类的配置：

- 对于实例化Bean的逻辑比较复杂，则比较适合用基于Java类配置的方式

简化Spring XML配置

在本套课程中我们深入和详细的学习了简化Spring配置文件的方法。你应当掌握了以下知识：

- Bean的自动装配
- 基于注解的配置
- 基于Java类的配置
- 不同配置方式的优劣

通过本课程的学习，你应该对Spring的配置文件的简化有了比较深入的了解，可以根据实际情况来选择不同的配置方式。如果你想继续提高，可以继续在极客学院学习Spring的其他相关课程。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台