# Spring学习大纲

第一部分： spring框架的概述以及spring中基于XML的IOC（反转控制）配置

第二部分：Spring中基于注解的IOC和IOC案例

第三部分：spring中的AOP和基于XML以及注解的AOP（面向切面编程）配置

第四部分：spring中的JDBC Temlate以及Spring事物控制

### Spring学习第二天——Spring注解实现IOC以及实例

在这一部分学习内容中，主要的内容分为两个部分，第一部分是用xml配置文件来完成数据库相关操作类的创建，第二部分则是通过注解的方法替换xml，来实现spring的ioc。此外，还补充了通过junit替换运行器来创建数据库连接容器。

**1、基于XML配置的IOC**

在这两个部分的内容中，使用DBAssit作为持久层解决方案，使用c3p0数据源。

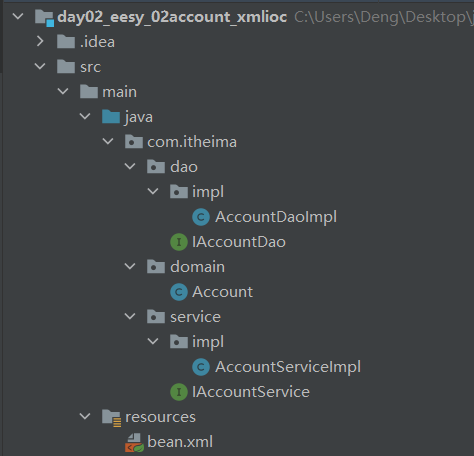
* 1. **必要的准备**

首先，由于使用了dbAssit等等工具，需要导入额外的jar包。



接着，需要为数据库操作的持久层、逻辑层以及实体类编写一些必要的类，如Account类作为实体类是被操作的对象；IAccountDao是持久层接口，声明CRUD操作；AccountDaoImpl类是持久层实际操作的类，实现CRUD方法，在这个类中有一个dbAssit类作为属性，来进行数据库连接。同时还有AccountServiceImpl类来封装持久层接口，进行数据库操作。

大致项目结构如下。



使用ioc完成CRUD操作所需要的几个类如上所述，接下来就是xml的配置部分。

* 1. **xml配置**
     1. **数据源的配置**

Java需要配置数据源来进行数据库的连接和访问，因此首先需要在xml中配置一个数据源类。具体代码如下所示。

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  
 <!--连接数据库的必备信息-->  
 <property name="driverClass" value="com.mysql.jdbc.Driver"></property>  
 <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/eesy"></property>  
 <property name="user" value="root"></property>  
 <property name="password" value="1234"></property>  
</bean>

数据源中包含了数据库连接的进本信息，如数据库用户名密码、连接池管理等等内容。

* + 1. **dbAssit的配置**

dbAssit是导入的jar包中的类，数据源是一个属性。dbAssit应该是用来负责数据库的连接与操作的（其实也可以用JDBCTemplate来代替），具体代码如下。

<bean id="dbAssit" class="comitheima.dao.dbassit.DBAssit">  
 <!--注入数据源-->  
 <property name="dataSource" ref="dataSource"></property>  
</bean>

配置完dbAssit后，就可以通过dbAssit来对数据库进行操作了。

* + 1. **Dao和Service的配置**

将dbAssit注入到accountDao中，就可以在持久层利用dbAssit进行数据库操作了。

<bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl">  
 <property name="dbAssit" ref="dbAssit"></property>  
</bean>

同样，将accountDao注入到Service中，因为Service才是逻辑层使用的对象。

<bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl">  
 <!-- 注入dao -->  
 <property name="accountDao" ref="accountDao"></property>  
</bean>

* 1. **代码实现**

由于Service类才是在逻辑层中被使用的对象，因此，通过反射创建accountService对象，就可以进行CRUD操作了。

ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml"); IAccountService as = ac.getBean("accountService",IAccountService.class);

在每次进行CRUD操作时，使用上述代码创建对象，即可进行CRUD操作。

此外，还可以使用Java的整合Junit，来自动创建访问数据库的类容器，而不需要手动代码创建。

**2、基于注解的IOC**

基于注解实现IOC，实际上是将注解写在类的定义中，用各种各样的注解实现IOC。

### 2.1 常用注解

**2.1.1 用于创建对象的注解**

@Component: 把当前类对象存入spring容器中,其属性如下:

value: 用于指定当前类的id. 不写时默认值是当前类名,且首字母改小写

@Controller: 将当前表现层对象存入spring容器中

@Service: 将当前业务层对象存入spring容器中

@Repository: 将当前持久层对象存入spring容器中

这些注解的作用相当于bean.xml中的<bean>标签

**注意：如果注解中有且只有一个属性要赋值时，且名称是 value，value 在赋值是可以不写**

@Controller,@Service,@Repository注解的作用和属性与@Component是一模一样的,可以相互替代,它们的作用是使三层对象的分别更加清晰.

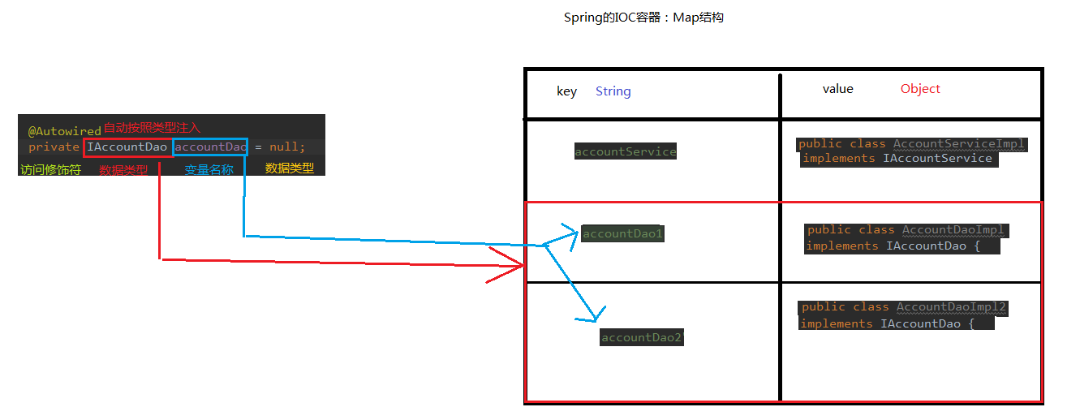
**2.1.2 用于数据注入的注解**

@Autowired: 自动按照成员变量类型注入.

**注入过程**

当spring容器中有且只有一个对象的类型与要注入的类型相同时,注入该对象.

当spring容器中**有多个对象类型与要注入的类型相同**时,使用要注入的**变量名作为bean的id**,在spring 容器查找,找到则注入该对象.找不到则报错.



**出现位置:** 既可以在变量上,也可以在方法上

**细节:** 使用注解注入时,set方法可以省略

@Qualifier: 在自动按照类型注入的基础之上,再按照bean的id注入.

1、出现位置: 既可以在变量上,也可以在方法上.注入变量时不能独立使用,必须和@Autowire一起使用; 注入方法时可以独立使用.

属性:value: 指定bean的id

@Resource: 直接按照bean的id注入,它可以独立使用.独立使用时相当于同时使用@Autowired和@Qualifier两个注解.

属性:name: 指定bean的id

@Value: 注入基本数据类型和String类型数据

属性:value: 用于指定数据的值,可以使用el表达式(${表达式})

**2.1.3 用于改变作用范围的注解**

这些注解的作用相当于bean.xml中的<bean>标签的scope属性.

@Scope: 指定bean的作用范围

属性:value:

用于指定作用范围的取值,"singleton" ,"prototype","request","session","globalsession"

**2.1.4 生命周期相关的注解**

这些注解的作用相当于bean.xml中的<bean>标签的init-method和destroy-method属性

1. @PostConstruct: 用于指定初始化方法
2. @PreDestroy: 用于指定销毁方法

# 2.2 spring的半注解配置

spring的注解配置可以与xml配置并存,也可以只使用注解配置。

在半注解配置下,spring容器仍然使用ClassPathXmlApplicationContext类从xml文件中读取IOC配置,同时在xml文件中告知spring创建容器时要扫描的包.

使用半注解模式时,上述实例中的beans.xml内容如下:

<!-- 告知spring在创建容器时要扫描的包 -->  
<context:component-scan

base-package="com.itheima">

</context:component-scan>

这样一来，所有没有在xml配置文件中配置的类，都可以通过扫描包与定义类时上面的注解来创建容器。

# 2.2 spring的纯注解配置

我们发现，现在的xml配置文件中有一句很关键的配置：

<!-- 告知spring在创建容器时要扫描的包 -->  
<context:component-scan

base-package="com.itheima">

</context:component-scan>

如果这个声明也能用注解配置，那么就离脱离 xml 文件又进了一步。

因此，在纯注解配置下,可以用配置类替代bean.xml,spring容器使用AnnotationApplicationContext类从spring配置类中读取IOC配置，此时，可以将xml文件彻底移除。

2.2.1纯注解配置下的注解

@Configuration: 用于指定当前类是一个spring配置类,当创建容器时会从该类上加载注解.获取容器时需要使用AnnotationApplicationContext(有@Configuration注解的类.class).

@ComponentScan: 指定spring在初始化容器时要扫描的包,作用和bean.xml 文件中<context:component-scan base-package="要扫描的包名"/>是一样的. 其属性如下:

basePackages: 用于指定要扫描的包,是value属性的别名

@Bean: 该注解只能写在方法上,表明使用此方法创建一个对象,并放入spring容器,其属性如下:

name: 指定此方法创建出的bean对象的id

细节: 使用注解配置方法时,如果方法有参数,Spring框架会到容器中查找有没有可用的bean对象,查找的方式与@Autowired注解时一样的.

@PropertySource: 用于加载properties配置文件中的配置.例如配置数据源时,可以把连接数据库的信息写到properties配置文件中,就可以使用此注解指定properties配置文件的位置,其属性如下:

value: 用于指定properties文件位置.如果是在类路径下,需要写上"classpath:"

@Import: 用于导入其他配置类.当我们使用@Import注解之后,有@Import注解的类就是父配置类,而导入的都是子配置类. 其属性如下:

value: 用于指定其他配置类的字节码

**2.2.2纯注解配置实现CRUD**

dao层：

@Repository("accountDao")

public class AccountDaoImpl implements IAccountDao {

@Autowired // 自动从spring容器中寻找QueryRunner类型对象注入给runner成员变量

private QueryRunner runner; // DBUtil对象,用来执行SQL语句

public List<Account> findAllAccount() {

// 功能实现...

}

public void saveAccount(Account account) {

// 功能实现...

}

public void deleteAccount(Integer accountId) {

// 功能实现...

}

}

可以看到，QueryRunner类从xml配置注入变成了注解注入。

SpringConfiguration类：

包config存放配置类，其中SpringConfiguration类为主配置类,内容如下:

@ComponentScan("com.itheima") //说明此类为配置类  
@Import(JdbcConfig.class) //指定初始化容器时要扫描的包  
@PropertySource("classpath:jdbcConfig.properties") //引入JDBC配置类  
public class SpringConfiguration {  
}

JDBCConfig类为JDBC配置类,内容如下:

@Configuration // 说明此类为配置类

@PropertySource("classpath:jdbc.properties") // 指定配置文件的路径,关键字classpath表示类路径

public class JdbcConfig {

@Value("${jdbc.driver}") // 为driver成员属性注入值,使用el表达式

private String driver;

@Value("${jdbc.url}") // 为url成员属性注入值,使用el表达式

private String url;

@Value("${jdbc.username}") // 为usernamer成员属性注入值,使用el表达式

private String username;

@Value("${jdbc.password}") // 为password成员属性注入值,使用el表达式

private String password;

// 创建DBUtils对象

@Bean(name="runner") // 此将函数返回的bean对象存入spring容器中,其id为runner

@Scope("prototype") // 说明此bean对象的作用范围为多例模式,以便于多线程访问

public QueryRunner createQueryRunner(@Qualifier("ds") DataSource dataSource){

// 为函数参数datasource注入id为ds的bean对象

return new QueryRunner(dataSource);

}

// 创建数据库连接池对象

@Bean(name="ds") // 此将函数返回的bean对象存入spring容器中,其id为ds

public DataSource createDataSource(){

try {

ComboPooledDataSource ds = new ComboPooledDataSource();

ds.setDriverClass(driver);

ds.setJdbcUrl(url);

ds.setUser(username);

ds.setPassword(password);

return ds;

}catch (Exception e){

throw new RuntimeException(e);

}

}

}

因此至此，我们可以引入两个配置类SpringConfiguration和JdbcConfig，再通过@Import引入JdbcConfig配置类，[再通过@PropertySource引入指定jdbc.properties](mailto:再通过@PropertySource引入指定jdbc.properties)的配置信息，可以将完全摆脱bean.xml了。

最后，通过AnnotationApplicationContext获取数据库连接工具类的容器。：

ApplicationContext ac = new AnnotationConfigApplicationContext(SpringConfiguration.class);

如此一来，就实现了完全使用注解的纯注解方法。

**2.3基于XML配置与基于注解配置的比较：**



基于注解的 spring IoC 配置中，bean 对象的特点和基于 XML 配置是一模一样的。

**3、【掌握】Spring整合Junit**

在上述的代码中，每次进行数据库操作都要先创建一个Service的类。

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

IAccountService as = ac.getBean("accountService",IAccountService.class);

junit 给了一个注解，可以用来替换掉它的运行器。只需要我们依靠 spring 框架，因为它提供了一个运行器，可以读取配置文件（或注解）来创建容器。我们只需要告诉它配置文件在哪就行了。

整体代码如下：

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations= {"classpath:bean.xml"})

public class AccountServiceTest

{

@Autowired

private IAccountService as ;

}

@RunWith注解用来替换原有运行器

@ContextConfiguration 注解：

locations 属性：用于指定配置文件的位置。如果是类路径下，需要用 classpath:表明

classes 属性：用于指定注解的类。当不使用 xml 配置时，需要用此属性指定注解类的位置。

如此一来，就能自动创建容器了。

**4、总结**

在这一部分学习内容中，主要的内容分为两个部分，第一部分是用xml配置文件来完成数据库相关操作类的创建，第二部分则是通过注解的方法替换xml，来实现spring的ioc，可以用一半注解，一般xml方式，也能完全用纯注解来取代xml。此外，还补充了通过junit替换运行器来自动读取配置文件创建容器。