Spring – day02

学习目标

（1）能够掌握spring中的常用注解

（2）能够了解spring中的新注解

（3）能够自主完成spring整合junit

学习指南

spring中ioc的常用注解

案例使用xml方式和注解方式实现单表的CRUD操作

改造基于注解的ioc案例，使用纯注解的方式实现

spring和Junit整合

# 第一堂课

**本节知识点**

01今日课果程内容介绍

02常用IOC注解按照作用分类

03用于创建的Component注解

04由Component衍生的注解

**本节目标**

知道注解被分为了几类

能够知道Component注解的作用

知道Component注解衍生出来哪3个注解

**视频时间：**

24分52秒

## 今日课果程内容介绍

### 知识概述

### 视频详情



### 总结与补充

spring第二天：spring基于注解的IOC以及IoC的案例

1、spring中ioc的常用注解

2、案例使用xml方式和注解方式实现单表的CRUD操作

持久层技术选择：dbutils

3、改造基于注解的ioc案例，使用纯注解的方式实现

spring的一些新注解使用

4、spring和Junit整合

### 课堂提问与练习

无

### 习题答案

无

## 常用IOC注解按照作用分类

### 知识概述

### 视频详情



### 总结与补充

基于注解的 IOC配置

学习基于注解的 IoC 配置，大家脑海里首先得有一个认知，即注解配置和 xml 配置要实现的功能都是一样

的，都是要降低程序间的耦合。只是配置的形式不一样。

关于实际的开发中到底使用xml还是注解，每家公司有着不同的使用习惯。所以这两种配置方式我们都需要掌

握。

我们在讲解注解配置时，采用上一章节的案例，把 spring 的 xml 配置内容改为使用注解逐步实现。

新建一个工程day02\_eesy\_01anno\_ioc导入spring-day01中day01\_eesy\_03spring的代码

注解的分类：



### 课堂提问与练习

无

### 习题答案

无

## 用于创建的Component注解

### 知识概述

### 视频详情



### 总结与补充

Component:

作用：用于把当前类对象存入spring容器中

属性：

value：用于指定bean的id。当我们不写时，它的默认值是当前类名，且首字母改小写。

<!--告知spring在创建容器时要扫描的包，配置所需要的标签不是在beans的约束中，而是一个名称为 context名称空间和约束中-->

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

### 课堂提问与练习

怎么让注解生效？

### 习题答案

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

## 由Component衍生的注解

### 知识概述

### 视频详情



### 总结与补充

Controller：一般用在表现层

Service：一般用在业务层

Repository：一般用在持久层

以上三个注解他们的作用和属性与Component是一模一样

他们三个是spring框架为我们提供明确的三层使用的注解，使我们的三层对象更加清晰

### 课堂提问与练习

无

### 习题答案

无

# 第二堂课

**本节知识点**

2.1 自动按照类型注入

2.2 用于注入数据的注解

2.3 改变作用范围以及和生命周期相关的注解

**本节目标**

能够知道哪个注解能按照类型注入

能够清楚按照名称注入的注解及注入数据的注解

能够清晰知道作用范围的注解

**视频时间：**

28分49秒

## 自动按照类型注入

### 知识概述

### 视频详情



### 总结与补充

用于注入数据的

他们的作用就和在xml配置文件中的bean标签中写一个<property>标签的作用是一样的

Autowired:

作用：自动按照类型注入。只要容器中有唯一的一个bean对象类型和要注入的变量类型匹配，就可以注入成功

如果ioc容器中没有任何bean的类型和要注入的变量类型匹配，则报错。

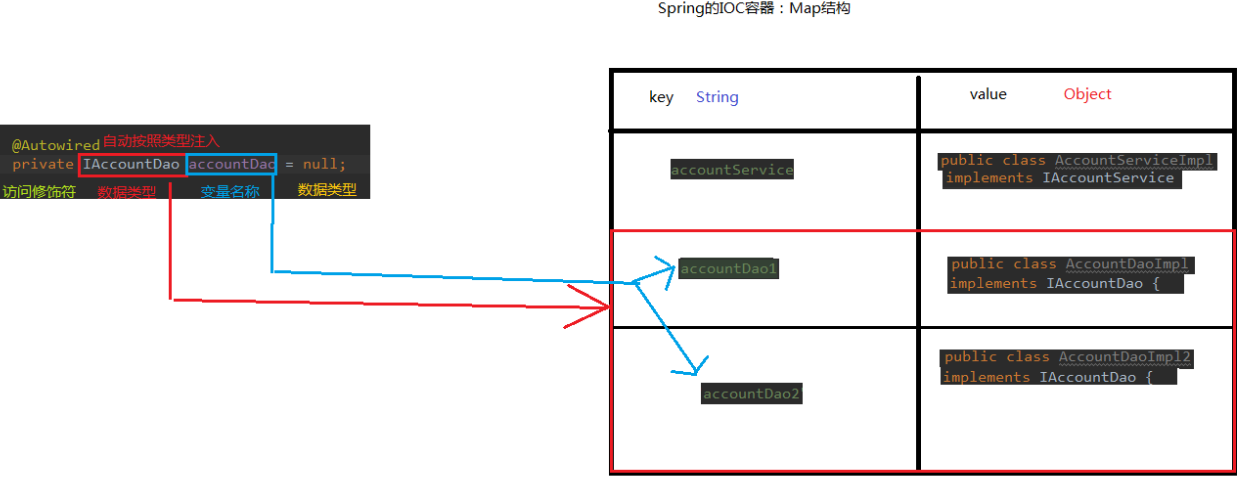
如果Ioc容器中有多个类型匹配时：

出现位置：

可以是变量上，也可以是方法上

细节：

在使用注解注入时，set方法就不是必须的了。



### 课堂提问与练习

按照类型属性注入用哪个注解？

### 习题答案

@Autowired

## 用于注入数据的注解

### 知识概述

### 视频详情



### 总结与补充

Qualifier:

作用：在按照类中注入的基础之上再按照名称注入。它在给类成员注入时不能单独使用。但是在给方法参数注入时可以（稍后我们讲）

属性：

value：用于指定注入bean的id。

Resource

作用：直接按照bean的id注入。它可以独立使用

属性：

name：用于指定bean的id。

以上三个注入都只能注入其他bean类型的数据，而基本类型和String类型无法使用上述注解实现。

另外，集合类型的注入只能通过XML来实现。

Value

作用：用于注入基本类型和String类型的数据

属性：

value：用于指定数据的值。它可以使用spring中SpEL(也就是spring的el表达式）

SpEL的写法：${表达式}

### 课堂提问与练习

无

## 改变作用范围以及和生命周期相关的注解

### 知识概述

### 视频详情



### 总结与补充

用于改变作用范围的

他们的作用就和在bean标签中使用scope属性实现的功能是一样的

Scope

作用：用于指定bean的作用范围

属性：

value：指定范围的取值。常用取值：singleton prototype

和生命周期相关 (了解)

他们的作用就和在bean标签中使用init-method和destroy-methode的作用是一样的

PreDestroy

作用：用于指定销毁方法

PostConstruct

作用：用于指定初始化方法

### 课堂提问与练习

无

### 习题答案

无

# 第三堂课

**本节知识点**

3.1 XMLIOC的案例-准备案例的必须代码

3.2 XMLIOC案例-编写spring的Ioc配置

3.3 测试基于XML的IOC案例

**本节目标**

能够编写案例的代码(xml方式)

**视频时间：**

32分03秒

## XMLIOC的案例-准备案例的必须代码

### 知识概述

### 视频详情



### 总结与补充

1. 创建一个新工程day02\_eesy\_02account\_xmlioc
2. 导入坐标pom.xml文件

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>5.0.2.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>5.0.2.RELEASE</version>  </dependency>  <dependency>  <groupId>commons-dbutils</groupId>  <artifactId>commons-dbutils</artifactId>  <version>1.4</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.6</version>  </dependency>  <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  </dependencies> |

1. 创建表和实体类

|  |
| --- |
| create table account(  id int primary key auto\_increment,  name varchar(40),  money float  )character set utf8 collate utf8\_general\_ci;  insert into account(name,money) values('aaa',1000);  insert into account(name,money) values('bbb',1000);  insert into account(name,money) values('ccc',1000); |

Account

|  |
| --- |
| /\*\*  \* 账户的实体类  \*/  public class Account implements Serializable {  private Integer id;  private String name;  private Float money;  } |

1. AccountServiceImpl

|  |
| --- |
| \*\*  \* 账户的业务层实现类  \*/  public class AccountServiceImpl implements IAccountService{  private IAccountDao accountDao;  public void setAccountDao(IAccountDao accountDao) {  this.accountDao = accountDao;  }  @Override  public List<Account> findAllAccount() {  return accountDao.findAllAccount();  }  @Override  public Account findAccountById(Integer accountId) {  return accountDao.findAccountById(accountId);  }  @Override  public void saveAccount(Account account) {  accountDao.saveAccount(account);  }  @Override  public void updateAccount(Account account) {  accountDao.updateAccount(account);  }  @Override  public void deleteAccount(Integer acccountId) {  accountDao.deleteAccount(acccountId);  }  } |

1. AccountDaoImpl

|  |
| --- |
| /\*\*  \* 账户的持久层实现类  \*/  public class AccountDaoImpl implements IAccountDao {  private QueryRunner runner;  public void setRunner(QueryRunner runner) {  this.runner = runner;  }  @Override  public List<Account> findAllAccount() {  try{  return runner.query("select \* from account",new BeanListHandler<Account>(Account.class));  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public Account findAccountById(Integer accountId) {  try{  return runner.query("select \* from account where id = ? ",new BeanHandler<Account>(Account.class),accountId);  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public void saveAccount(Account account) {  try{  runner.update("insert into account(name,money)values(?,?)",account.getName(),account.getMoney());  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public void updateAccount(Account account) {  try{  runner.update("update account set name=?,money=? where id=?",account.getName(),account.getMoney(),account.getId());  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public void deleteAccount(Integer accountId) {  try{  runner.update("delete from account where id=?",accountId);  }catch (Exception e) {  throw new RuntimeException(e);  }  }  } |

### 课堂提问与练习

无

### 习题答案

无

## XMLIOC案例-编写spring的Ioc配置

### 知识概述

### 视频详情



### 总结与补充

bean.xml

|  |
| --- |
| <?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">  <!-- 配置Service -->  <bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl">  <!-- 注入dao -->  <property name="accountDao" ref="accountDao"></property>  </bean>  <!--配置Dao对象-->  <bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl">  <!-- 注入QueryRunner -->  <property name="runner" ref="runner"></property>  </bean>  <!--配置QueryRunner-->  <bean id="runner" class="org.apache.commons.dbutils.QueryRunner" scope="prototype">  <!--注入数据源-->  <constructor-arg name="ds" ref="dataSource"></constructor-arg>  </bean>  <!-- 配置数据源 -->  <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>  </beans> |

### 课堂提问与练习

无

### 习题答案

无

## 测试基于XML的IOC案例

### 知识概述

### 视频详情



### 总结与补充

测试代码：

|  |
| --- |
| /\*\*  \* 使用Junit单元测试：测试我们的配置  \*/  @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(locations = "classpath:bean.xml")  public class AccountServiceTest {  @Autowired  private IAccountService as;  @Test  public void testFindAll() {  //3.执行方法  List<Account> accounts = as.findAllAccount();  for(Account account : accounts){  System.out.println(account);  }  }  @Test  public void testFindOne() {  //3.执行方法  Account account = as.findAccountById(1);  System.out.println(account);  }  @Test  public void testSave() {  Account account = new Account();  account.setName("test");  account.setMoney(12345f);  //3.执行方法  as.saveAccount(account);  }  @Test  public void testUpdate() {  //3.执行方法  Account account = as.findAccountById(4);  account.setMoney(23456f);  as.updateAccount(account);  }  @Test  public void testDelete() {  //3.执行方法  as.deleteAccount(4);  }  } |

### 课堂提问与练习

无

### 习题答案

无

# 第四堂课

**本节知识点**

4.1 注解IOC案例-把自己编写的类使用注解配置

4.2 spring的新注解-Configuration和ComponentScan

4.3 spring的新注解-Bean

**本节目标**

能够改造xml文件配置方式的项目为注解方式

掌握Configuration和ComponentScan的作用

能够掌握Bean注解的作用

**视频时间：**

27分59秒

## 注解IOC案例-把自己编写的类使用注解配置

### 知识概述

### 视频详情



### 总结与补充

1.新建项目day02\_eesy\_03account\_annoioc

2.pom.xml

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>5.0.2.RELEASE</version>  </dependency>  <dependency>  <groupId>commons-dbutils</groupId>  <artifactId>commons-dbutils</artifactId>  <version>1.4</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.6</version>  </dependency>  <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.10</version>  </dependency>  </dependencies> |

3. AccountServiceImpl

|  |
| --- |
| /\*\*  \* 账户的业务层实现类  \*/  @Service("accountService")  public class AccountServiceImpl implements IAccountService{  @Autowired  private IAccountDao accountDao;  @Override  public List<Account> findAllAccount() {  return accountDao.findAllAccount();  }  @Override  public Account findAccountById(Integer accountId) {  return accountDao.findAccountById(accountId);  }  @Override  public void saveAccount(Account account) {  accountDao.saveAccount(account);  }  @Override  public void updateAccount(Account account) {  accountDao.updateAccount(account);  }  @Override  public void deleteAccount(Integer acccountId) {  accountDao.deleteAccount(acccountId);  }  } |

4. AccountDaoImpl

|  |
| --- |
| /\*\*  \* 账户的持久层实现类  \*/  @Repository("accountDao")  public class AccountDaoImpl implements IAccountDao {  @Autowired  private QueryRunner runner;  @Override  public List<Account> findAllAccount() {  try{  return runner.query("select \* from account",new BeanListHandler<Account>(Account.class));  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public Account findAccountById(Integer accountId) {  try{  return runner.query("select \* from account where id = ? ",new BeanHandler<Account>(Account.class),accountId);  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public void saveAccount(Account account) {  try{  runner.update("insert into account(name,money)values(?,?)",account.getName(),account.getMoney());  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public void updateAccount(Account account) {  try{  runner.update("update account set name=?,money=? where id=?",account.getName(),account.getMoney(),account.getId());  }catch (Exception e) {  throw new RuntimeException(e);  }  }  @Override  public void deleteAccount(Integer accountId) {  try{  runner.update("delete from account where id=?",accountId);  }catch (Exception e) {  throw new RuntimeException(e);  }  }  } |

5．测试类

|  |
| --- |
| \*\*  \* 使用Junit单元测试：测试我们的配置  \*/  public class AccountServiceTest {  @Test  public void testFindAll() {  //1.获取容易  ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");  //2.得到业务层对象  IAccountService as = ac.getBean("accountService",IAccountService.class);  //3.执行方法  List<Account> accounts = as.findAllAccount();  for(Account account : accounts){  System.out.println(account);  }  }  @Test  public void testFindOne() {  //1.获取容易  ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");  //2.得到业务层对象  IAccountService as = ac.getBean("accountService",IAccountService.class);  //3.执行方法  Account account = as.findAccountById(1);  System.out.println(account);  }  @Test  public void testSave() {  Account account = new Account();  account.setName("test");  account.setMoney(12345f);  //1.获取容易  ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");  //2.得到业务层对象  IAccountService as = ac.getBean("accountService",IAccountService.class);  //3.执行方法  as.saveAccount(account);  }  @Test  public void testUpdate() {  //1.获取容易  ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");  //2.得到业务层对象  IAccountService as = ac.getBean("accountService",IAccountService.class);  //3.执行方法  Account account = as.findAccountById(4);  account.setMoney(23456f);  as.updateAccount(account);  }  @Test  public void testDelete() {  //1.获取容易  ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");  //2.得到业务层对象  IAccountService as = ac.getBean("accountService",IAccountService.class);  //3.执行方法  as.deleteAccount(4);  }  } |

### 课堂提问与练习

无

### 习题答案

无

## spring的新注解-Configuration和ComponentScan

### 知识概述

### 视频详情



### 总结与补充

创建：SpringConfiguration.java 该类是一个配置类，它的作用和bean.xml是一样的

说明两个注解：

@Configuration

作用：指定当前类是一个配置类

细节：当配置类作为AnnotationConfigApplicationContext对象创建的参数时，该注解可以不写。

@ComponentScan("com.itheima")

作用：用于通过注解指定spring在创建容器时要扫描的包

属性：

value：它和basePackages的作用是一样的，都是用于指定创建容器时要扫描的包。

我们使用此注解就等同于在xml中配置了:

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

### 课堂提问与练习

无

### 习题答案

无

## spring的新注解-Bean

### 知识概述

### 视频详情



### 总结与补充

|  |
| --- |
| public class SpringConfiguration {  /\*\*  \* 用于创建一个QueryRunner对象  \* @param dataSource  \* @return  \*/  @Bean(name="runner")  public QueryRunner createQueryRunner(@Qualifier("ds2") DataSource dataSource){  return new QueryRunner(dataSource);  }  /\*\*  \* 创建数据源对象  \* @return  \*/  @Bean(name="ds2")  public DataSource createDataSource(){  try {  ComboPooledDataSource ds = new ComboPooledDataSource();  ds.setDriverClass("com.mysql.jdbc.Driver");  ds.setJdbcUrl("jdbc:mysql://localhost:3306/eesy");  ds.setUser("root");  ds.setPassword("1234");  return ds;  }catch (Exception e){  throw new RuntimeException(e);  }  }  } |

Bean

作用：用于把当前方法的返回值作为bean对象存入spring的ioc容器中

属性:

name:用于指定bean的id。当不写时，默认值是当前方法的名称

细节：

当我们使用注解配置方法时，如果方法有参数，spring框架会去容器中查找有没有可用的bean对象。

查找的方式和Autowired注解的作用是一样的

### 课堂提问与练习

无

### 习题答案

无

# 第五堂课

**本节知识点**

5.1 AnnotationConfigApplicationContext的使用

5.2 spring的新注解-Import

5.3 spring的新注解-PropertySource

**本节目标**

会使用AnnotationConfigApplicationContext进行测试

能够使用import注解

spring的新注解-PropertySource

**视频时间：**

28分09秒

## AnnotationConfigApplicationContext的使用

### 知识概述

### 视频详情



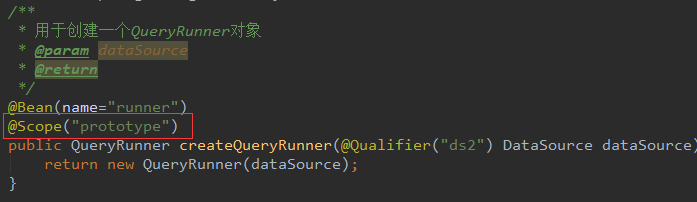
### 总结与补充

修改现有的测试类，删除之前的bean.xml文件

使用一下代码进行测试

ApplicationContext ac = new AnnotationConfigApplicationContext(SpringConfiguration.class);

在创建创建一个QueryRunner对象的方法上加上@Scope("prototype")，让其为多例



### 课堂提问与练习

无

### 习题答案

无

## spring的新注解-Import

### 知识概述

### 视频详情



### 总结与补充

把所有的数据连接配置相关的东西从SpringConfiguration类中挪到了createDataSource类中

作用：用于导入其他的配置类

属性：

value：用于指定其他配置类的字节码。

当我们使用Import的注解之后，有Import注解的类就父配置类，而导入的都是子配置类

修改后的SpringConfiguration

|  |
| --- |
| @ComponentScan("com.itheima")  @Import(JdbcConfig.class)  @PropertySource("classpath:jdbcConfig.properties")  public class SpringConfiguration {  } |

新增的

|  |
| --- |
| /\*\*  \* 和spring连接数据库相关的配置类  \*/  public class JdbcConfig {  /\*\*  \* 用于创建一个QueryRunner对象  \* @param dataSource  \* @return  \*/  @Bean(name="runner")  @Scope("prototype")  public QueryRunner createQueryRunner(@Qualifier("ds2") DataSource dataSource){  return new QueryRunner(dataSource);  }  /\*\*  \* 创建数据源对象  \* @return  \*/  @Bean(name="ds2")  public DataSource createDataSource(){  try {  ComboPooledDataSource ds = new ComboPooledDataSource();  ds.setDriverClass("com.mysql.jdbc.Driver");  ds.setJdbcUrl("jdbc:mysql://localhost:3306/eesy");  ds.setUser("root");  ds.setPassword("1234");  return ds;  }catch (Exception e){  throw new RuntimeException(e);  }  } |

### 课堂提问与练习

无

### 习题答案

无

## spring的新注解-PropertySource

### 知识概述

### 视频详情



### 总结与补充

创建jdbcConfig.properties文件

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/eesy

jdbc.username=root

jdbc.password=1234

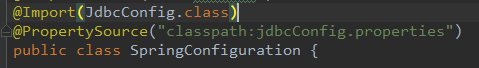
PropertySource

作用：用于指定properties文件的位置

属性：

value：指定文件的名称和路径。

关键字：classpath，表示类路径下



修改

|  |
| --- |
| \*\*  \* 和spring连接数据库相关的配置类  \*/  public class JdbcConfig {  @Value("${jdbc.driver}")  private String driver;  @Value("${jdbc.url}")  private String url;  @Value("${jdbc.username}")  private String username;  @Value("${jdbc.password}")  private String password;  /\*\*  \* 创建数据源对象  \* @return  \*/  @Bean(name="ds2")  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);  }  } |

思考：

注解与xml方式的选择？

哪种选择更方便就使用哪种方式来配置

### 课堂提问与练习

无

### 习题答案

无

# 第六堂课

**本节知识点**

6.1 Qualifier注解的另一种用法

6.2 spring整合Junit

**本节目标**

**视频时间：**

## Qualifier注解的另一种用法

### 知识概述

### 视频详情



### 总结与补充

@Qualifier("ds2")的用法，可以在方法的形参中指定你要注入的数据源名称

|  |
| --- |
| /\*\*  \* 和spring连接数据库相关的配置类  \*/  public class JdbcConfig {  @Value("${jdbc.driver}")  private String driver;  @Value("${jdbc.url}")  private String url;  @Value("${jdbc.username}")  private String username;  @Value("${jdbc.password}")  private String password;  /\*\*  \* 用于创建一个QueryRunner对象  \* @param dataSource  \* @return  \*/  @Bean(name="runner")  @Scope("prototype")  public QueryRunner createQueryRunner(@Qualifier("ds2") DataSource dataSource){  return new QueryRunner(dataSource);  }  /\*\*  \* 创建数据源对象  \* @return  \*/  @Bean(name="ds2")  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);  }  }  @Bean(name="ds1")  public DataSource createDataSource1(){  try {  ComboPooledDataSource ds = new ComboPooledDataSource();  ds.setDriverClass(driver);  ds.setJdbcUrl("jdbc:mysql://localhost:3306/eesy02");  ds.setUser(username);  ds.setPassword(password);  return ds;  }catch (Exception e){  throw new RuntimeException(e);  }  }  } |

### 课堂提问与练习

无

### 习题答案

无

## spring整合Junit问题分析

### 知识概述

### 视频详情



### 总结与补充

问题分析：

1、应用程序的入口

main方法

2、junit单元测试中，没有main方法也能执行

junit集成了一个main方法

该方法就会判断当前测试类中哪些方法有 @Test注解

junit就让有Test注解的方法执行

3、junit不会管我们是否采用spring框架

在执行测试方法时，junit根本不知道我们是不是使用了spring框架

所以也就不会为我们读取配置文件/配置类创建spring核心容器

4、由以上三点可知

当测试方法执行时，没有Ioc容器，就算写了Autowired注解，也无法实现注入

### 课堂提问与练习

无

### 习题答案

无

## spring整合Junit完成

### 知识概述

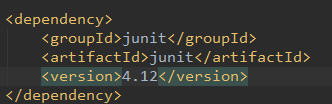
### 视频详情



### 总结与补充

**Spring整合junit的配置**

1导入spring整合junit的jar(坐标)



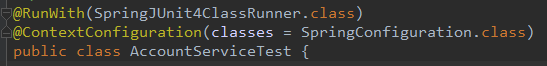
2、使用Junit提供的一个注解把原有的main方法替换了，替换成spring提供的 @Runwith

3、告知spring的运行器，spring和ioc创建是基于xml还是注解的，并且说明位置

@ContextConfiguration

locations：指定xml文件的位置，加上classpath关键字，表示在类路径下

classes：指定注解类所在地位置



特别注意：当我们使用spring 5.x版本的时候，要求junit的jar必须是4.12及以上

### 课堂提问与练习

无

### 习题答案

无