1.引言

前面花大量内容，重点学习了 Spring入门 的一些思想，以及简单的学习了 IOC 基础 以及基于XML的配置方式和注解方式。下面就来完成一个对单表进行 CURD 的案例，加深IOC的理解。

本篇文章只针对使用，具体详细的内容可以去看《Spring框架内专题(三)-Spring框架之DI依赖注入.md》。

2.准备代码与环境

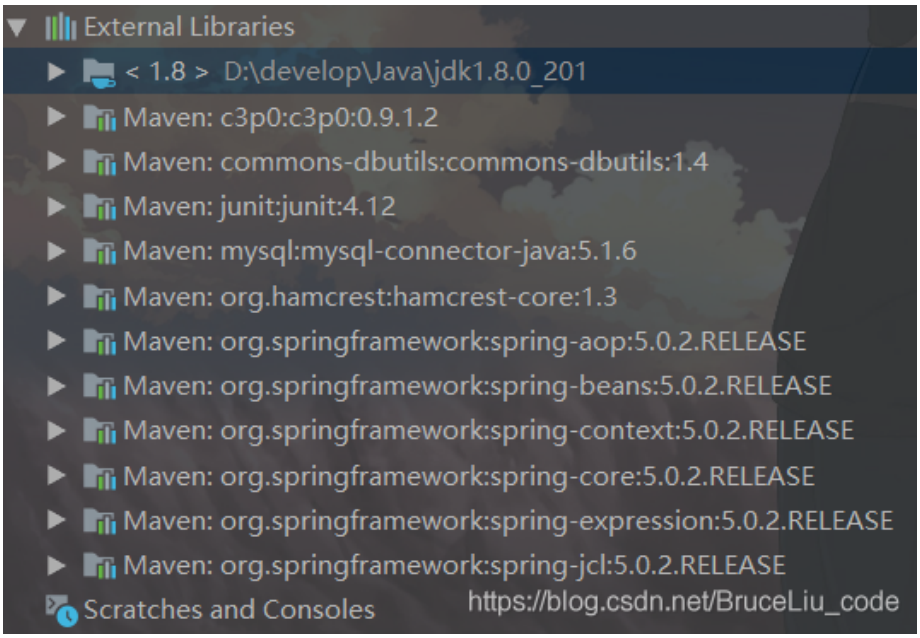
2.1.添加必要的依赖

- spring-context
- mysql-connector-java
- c3p0（数据库连接池）
- commons-dbutils（简化JDBC的工具）
- junit （单元自测）

> *说明：由于我这里创建的是一个Maven项目，所以在这里修改pom.xml添加一些必要的依赖坐标就可以。如果创建时没有使用依赖的朋友，去下载我们所需要的jar 包导入就可以了*

```
<dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.2.RELEASE</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>commons-dbutils</groupId>
            <artifactId>commons-dbutils</artifactId>
            <version>1.4</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
</dependencies>
```

简单看一下，spring核心的一些依赖，以及数据库相关的依赖等就都导入进来了。



2.2.创建账户表以及实体

A：创建 Account 表

```
-- ----------------------------
-- Table structure for account
-- ----------------------------
CREATE TABLE `account`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(32),
  `balance` float,
  PRIMARY KEY (`id`)
)
```

B.创建 Account 类

没什么好说的，对应着我们的表创出实体

```
public class Account implements Serializable {
    private  Integer id;
    private String name;
    private Float balance;
    ......补充 get set toString 方法
```

2.3.创建Service以及Dao

A：AccountService 接口

```
public interface AccountService {

    void add(Account account);

    void delete(Integer accpuntId);

    void update(Account account);

    List<Account> findAll();

    Account findById(Integer accountId);
}
```

B：AccountServiceImpl 实现类

```
public class AccountServiceImpl implements AccountService
{

    private AccountDao accountDao;

    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    public void add(Account account) {
        accountDao.addAccount(account);
    }

    public void delete(Integer accpuntId) {
        accountDao.deleteAccount(accpuntId);
    }

    public void update(Account account) {
        accountDao.updateAccount(account);
    }

    public List<Account> findAll() {
        return accountDao.findAllAccount();
    }

    public Account findById(Integer accountId) {
        return accountDao.findAccountById(accountId);
    }
}
```

C：AccountDao 接口

```
public interface AccountDao {

    void addAccount(Account account);

    void deleteAccount(Integer accountId);

    void updateAccount(Account account);

    List<Account> findAllAccount();

    Account findAccountById(Integer accountId);
}
```

D：AccountDaoImpl 实现类

由于今天要完成的是一个增删改查的操作，所以我们引入了 DBUtils 这样一个操作数据库的工具，它的作用就是封装代码，达到简化 JDBC 操作的目的，由于以后整合 SSM 框架的时候，持久层的事情就可以交给 MyBatis 来做，而今天我们重点还是讲解 Spring 中的知识，所以这部分会用就可以了，重点看 XML 与 注解 两种配置方式

用到的内容基本讲解：

- `QueryRunner` 提供对 sql 语句进行操作的 API （insert delete update）
- `ResultSetHander` 接口，定义了查询后，如何封装结果集（仅提供了我们用到的）

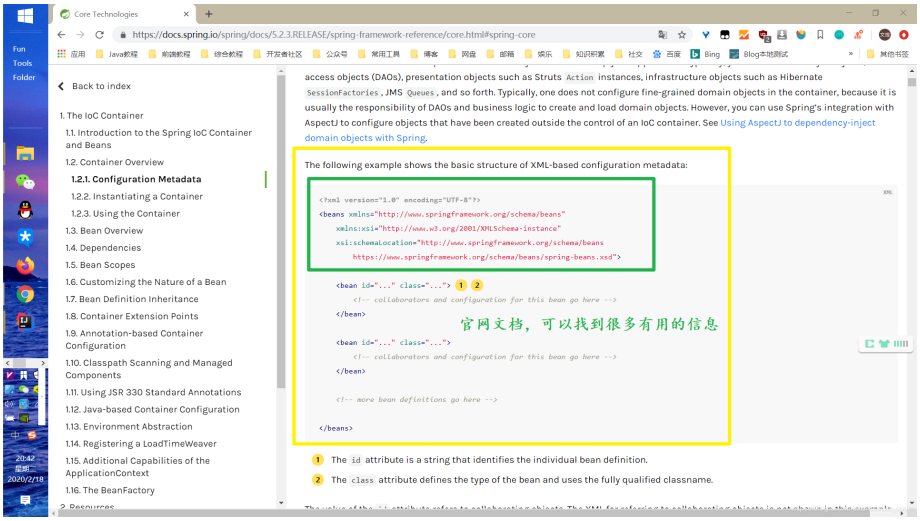
```
BeanHander：将结果集中的第一条记录封装到指定的 JavaBean 中
BeanListHandler：将结果集中的所有记录封装到指定的 JavaBean 中，并且将每一个 JavaBean封装到 List 中去
```

```
public class AccountDaoImpl implements AccountDao {


    private QueryRunner runner;

    public void setRunner(QueryRunner runner) {
        this.runner = runner;
    }

    public void addAccount(Account account) {
        try {
            runner.update("insert into
account(name,balance)values(?,?)", account.getName(),
account.getBalance());
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

```
    public void updateAccount(Account account) {
        try {
            runner.update("update account set
name=?,balance=? where id=?", account.getName(),
account.getBalance(), account.getId());
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public void deleteAccount(Integer accountId) {
        try {
            runner.update("delete from account where
id=?", accountId);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public List<Account> findAllAccount() {
        try {
            return runner.query("select * from account",
new BeanListHandler<Account>(Account.class));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    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);
        }
    }
}
```

3.XML配置方式

在这里有两基本的方式，一是通过构造函数注入，另一种就是通过Set注入，实际上所做的就是，使用类的构造函数或者Set给成员变量进行赋值，但特别的是，这里是通过配置，使用 Spring 框架进行注入首先就是头部的依赖信息，顺便提一句，当然我们可以去官网查找贴过来.



先把针对上面功能的具体配置代码贴出来

```
<?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="cn.ideal.service.impl.AccountServiceImpl">
        <property name="accountDao" ref="accountDao">
</property>
    </bean>

    <!--配置Dao-->
    <bean id="accountDao"
class="cn.ideal.dao.impl.AccountDaoImpl">
        <property name="runner" ref="runner"></property>
    </bean>

    <!--配置 QueryRunner-->
    <bean id="runner"
class="org.apache.commons.dbutils.QueryRunner">
        <!--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">
        <property name="driverClass"
value="com.mysql.jdbc.Driver"></property>
```

```
        <property name="jdbcUrl"
value="jdbc:mysql://localhost:3306/ideal_spring">
</property>
        <property name="user" value="root"></property>
        <property name="password" value="root99">
</property>

    </bean>
</beans>
```

3.1.Set方式

顾名思义，就是通过去找你给出对应的 Set 方法，然后对成员变量进行赋值，先看下类中的代码

```
public class AccountServiceImpl implements AccountService
{
    //成员
    private AccountDao accountDao;
    //Set方法
    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    ..... 下面是增删改查的方法
}
```

这是 bean.xml 中的配置

```
<!--配置Service-->
<bean id="accountService"
class="cn.ideal.service.impl.AccountServiceImpl">
    <property name="accountDao" ref="accountDao">
</property>
</bean>
```

然后 property 配置的过程中，有一些属性需要说一下

- name：与成员变量名无关，与set方法后的名称有关，例如 setAccountDao() 获取到的就是accountDao，并且已经小写了开头
- value：这里可以写基本数据类型和 String
- ref：这里可以引入另一个bean，帮助我们给其他类型赋值 （例如这里就通过 ref 引入了下面 id 值为accountDao的 bean）
  当然，以后可能会见到一种方式就是 使用 p名称空间注入数据 (本质还是set)

头部中需要修改引入这一句

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

直接拿以前一个例子：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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="accountService"
    class="cn.ideal.service.impl.AccountServiceImpl"
p:name="汤姆" p:age="21" p:birthday-ref="nowdt"/>
<bean id="nowdt" class="java.util.Date"></bean>
</beans>
```

3.2.构造函数方式

下面就是使用构造函数的一种方式，这一种的前提就是：类中必须提供一个和参数列表相对应的构造函数

由于我们选择的是 DBUtils 这样一个工具，而它为我们提供了两种构造函数，即带参和无参，所以我们可以在其中注入数据源，也可以使得每一条语句都独立事务

还有一点需要说明的就是：我们下面的数据源使用了 c3p0 这只是一种选择方式，并不是一定的，是因为使用 DBUtils 的时候需要手动传递一个 Connection 对象！

```
<!--配置 QueryRunner-->
<bean id="runner"
class="org.apache.commons.dbutils.QueryRunner">
    <!--注入数据源-->
    <constructor-arg name="ds" ref="dataSource">
</constructor-arg>
</bean>
```

来说一下所涉及到的标签：

- constructor-arg（放在 bean 标签内） 再说一说其中的一些属性值
- 给谁赋值：
  `index`：指定参数在构造函数参数列表的索引位置
  `type`：指定参数在构造函数中的数据类型
  `name`：指定参数在构造函数中的名称（更常用）
- 赋什么值：
  `value`：这里可以写基本数据类型和 String
  `ref`：这里可以引入另一个bean，帮助我们给其他类型赋值

3.3.测试代码

```
public class AccountServiceTest {
```

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

    @Test
    public void testAdd(){
        Account account = new Account();
        account.setName("jack");
        account.setBalance(1000f);
        as.add(account);
    }

    @Test
    public void testUpdate(){
        Account account = as.findById(4);
        account.setName("杰克");
        account.setBalance(1500f);
        as.update(account);
    }

    @Test
    public void testFindAll(){
        List<Account> list = as.findAll();
        for(Account account : list) {
            System.out.println(account);
        }
    }

    @Test
    public void testDelete(){
        as.delete(4);
    }
}
```

3.4.执行效果

添加，修改（包含了查询指定id），删除

查询所有

```
Account{id=1, name='张三', balance=1000.0}
Account{id=2, name='李四', balance=3000.0}
Account{id=3, name='王五', balance=2000.0}
```

4.注解配置方式

首先，我们先将上面的例子使用注解来实现一下，再来具体的讲解：

4.1.改造原程序为注解配置

首先需要为 Dao 和 Service 的实现类中 添加注解

```
@Service("accountService")
public class AccountServiceImpl implements AccountService
{
    @Autowired
    private AccountDao accountDao;

    下面的原封不动
}
```

```
@Repository("accountDao")
public class AccountDaoImpl implements AccountDao {
    @Autowired
    private QueryRunner runner;

    下面的原封不动
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"

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

http://www.springframework.org/schema/context/spring-
context.xsd">
    <!--开启扫描-->
    <context:component-scan base-package="cn.ideal">
</context:component-scan>

    <!--配置 QueryRunner-->
    <bean id="runner"
class="org.apache.commons.dbutils.QueryRunner">
        <!--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">
        <property name="driverClass"
value="com.mysql.jdbc.Driver"></property>
        <property name="jdbcUrl"
value="jdbc:mysql://localhost:3306/ideal_spring">
</property>
        <property name="user" value="root"></property>
        <property name="password" value="root99">
</property>
    </bean>
</beans>
```

到这里，一个最基本的注解改造就完成了，大家可以用前面的测试类进行一下测试
下面我们回顾说一下注解配置相关的知识!

4.2.常用注解

A：创建对象

@Component

- 让Spring 来管理资源，相当于XML 中配置一个 bean
- 可以在括号内指定 value 值，即指定 bean 的 id ，如果不指定就会默认的使用当前类的类名
- 如果注解中只有一个value属性要赋值，value可以不写，直接写名称，如上面例子中

@Controller @Service @Repository

对于创建对象的注解，Spring 还提供了三种更加明确的说法，作用是完全相同的，但是针对不同的场景起了不同的叫法罢了

- @Controller：一般用于表现层
- @Service：一般用于业务层
- @Repository：一般用于持久层

B：注入数据

@Autowired
自动按类型注入，相当于XML 中配置一个 bean `<property name="" ref="">` 或者 `<property name="" value="">`

容器中有一个唯一的 bean 对象类型和注入的变量类型一致，则注入成功

```
@Autowired
private AccountDao accountDao;

@Repository("accountDao")
public class AccountDaoImpl implements AccountDao {......}
```

比如上面的例子，Spring的IOC中容器是一个Map的结构，字符串“accountDao”以及这个可以认为是 AccountDao 类型的 AccountDaoImpl 类就被以键值对的形式存起来，被注解 @Autowired的地方，会直接去容器的 value 部分去找 AccountDao 这个类型的类

当 IoC 中匹配到了多个符合的，就会根据变量名去找，找不到则报错：例如下面，根据 AccountDao类型匹配到了两个类，所以根据变量名去找找到了 AccountDaoImplA 这个类

```
@Autowired
private AccountDao accountDaoA;

@Repository("accountDaoA")
public class AccountDaoImplA implements AccountDao
{......}

@Repository("accountDaoB")
public class AccountDaoImplB implements AccountDao
{......}
```

可以对类的成员变量、方法以及构造函数进行标注，完成自动装配，使用此注解可以省略 set 方法

@Qualifier

- 在自动按类型注入的基础之上，按照 Bean 的 id 注入，给字段注入的时候不能够单独使用，需要配合上面的 @Autiwire 使用，但是给方法参数注入的时候，可以独立使用
- 使用时：value 值指定 bean 的 id它有时候必须配合别的注解使用，有没有一个标签可以解决这个问题呢？答案就是 @Resource

@Resource

- 直接按照 bean 的 id 注入，不过只能注入其他 bean 类型
- 使用时：name 值指定 bean 的 id

> *前面三个都是用来注入其他的bean 类型的数据，下面来说一说，基本类型以及String的实现(特别说明：集合类型的注入只能通过 XML 来实现)*

@Value

- 这个注解就是用来注入基本数据类型和 String 类型数据的
- 使用时：value 属性用于指定值

C：改变作用范围

@Scope

- 指定 bean 的作用范围 相当于XML 中配置一个 <bean id="" class="" scope
- 使用时：value 属性用于指定范围的值（singleton prototype request session globalsession）

D：生命周期相关

- 相当于：`<bean id="" class="" init-method="" destroy-method="" />`

@PostConstruct

- 指定初始化方法

@PreDestroy

- 指定销毁方法

5.XML和注解的对比与选择

5.1.优缺点

一般来说，我们两种配置方式都是有人使用的，不过我个人更习惯使用注解的方式

XML：

- 类之间的松耦合关系，扩展性强，利于更换修改
- 对象之间的关系清晰明了

注解：

- 简化配置，并且使用起来也容易，效率会高一些
- 在类中就能找对配置，清晰明了
- 类型安全

5.2.两者对比

	XML配置	注解配置
创建对象	`<bean id="" class="">`	@Controller @Service @Repository@Component
指定名称	通过 id 或者 name 值指定	@Controller("指定的名称")
注入数据象	`<property name="" ref="">`	@Autowired @Qualifier @Resource @Value

	XML配置	注解配置
作用范围	`<bean id="" class="" scope>`	@Scope
生命周期	`<bean id="" class="" init-method="" destroy-method=""/>`	@PostConstruct @PreDestroy

6.补充新注解

为什么要补充新注解呢？ 在我们使用注解时，在书写代码时，简化了很多，但是我们在 bean.xml 文件中 仍然需要 开启扫描、 进行配置QueryRunner 以及 数据源，如何彻底摆脱 xml 配置全面使用注解呢？

这也就是我们将要补充的几个新注解，作用就是让我们全面使用注解进行开发

6.1.配置类注解

@Configuration

- 指定当前类是 **spring** 的一个配置类，相当于 **XML中的 bean.xml** 文件
- 获取容器时需要使用下列形式

```
private ApplicationContext ac = new
AnnotationConfigApplicationContext(SpringConfiguration.cla
ss);
//这里的SpringConfiguration类是我们自定义的，被Configuration注
解注释的类
```

依旧使用上方的 CURD 的案例代码进行修改，首先与cn同级创建了一个名为 config 的包，然后编写一个名为 SpringConfiguration 的类，当然实际上这两个名字无所谓的，添加注解

```
@Configuration
public class SpringConfiguration {
}
```

6.2.指定扫描包注解

@ComponentScan

@Configuration 相当于已经帮我们把 bean.xml 文件创立好了，按照我们往常的步骤，应该指定扫描的包了，这也就是我们这个注解的作用

- 指定 spring 在初始化容器时要扫描的包，在 XML 中相当于：

```
<!--开启扫描-->
<context:component-scan base-package="cn.ideal">
</context:component-scan>
```

- 其中 basePackages 用于指定扫描的包，和这个注解中value属性的作用是一致的

具体使用：

```
@Configuration
@ComponentScan("cn.ideal")
public class SpringConfiguration {
}
```

6.3.创建对象

@Bean

写好了配置类，以及指定了扫描的包，下面该做的就是配置 QueryRunner 以及数据源作为一个Bean实例，在 XML 中我们会通过书写 bean 标签来配置，而 Spring 为我们提供了 @Bean 这个注解来替代原来的标签

- 将注解写在方法上（只能是方法），也就是代表用这个方法创建一个对象，然后放到 Spring 的容器中去
- 通过 name 属性 给这个方法指定名称，也就是我们 XML 中 bean 的 id

具体使用：

```
package config;

import com.mchange.v2.c3p0.ComboPooledDataSource;
import org.apache.commons.dbutils.QueryRunner;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;

import javax.sql.DataSource;

//Configuration注解的Lite模式，在这种模式下JdbcConfig类中定义的Bean示例之间不能相互注入
//这里可以加上一个@Configuration注解
public class JdbcConfig {

    /**
     * 创建一个 QueryRunner对象
     * @param dataSource
     * @return
     */
    @Bean(name = "runner")
    public QueryRunner creatQueryRunner(DataSource dataSource){
        return  new QueryRunner(dataSource);
```

```
    }


    /**
     * 创建数据源，并且存入spring
     * @return
     */
    @Bean(name = "dataSource")
    public DataSource createDataSource() {
        try {
            ComboPooledDataSource ds = new
ComboPooledDataSource();
            ds.setUser("root");
            ds.setPassword("1234");
            ds.setDriverClass("com.mysql.jdbc.Driver");
            ds.setJdbcUrl("jdbc:mysql:///spring_day02");
            return ds;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

6.4.配置 properties文件

@PropertySource

上面在创建数据源的时候，都是直接把配置信息写死了，如果想要使用 properties 进行内容的配置，在这时候就需要，使用@PropertySource 这个注解

- 用于加载 .properties 文件中的配置
- value [] 指定 properties 文件位置，在类路径下，就需要加上 classpath

```
@Configuration
@ComponentScan("cn.ideal")
@PropertySource("classpath:jdbcConfig.properties")
public class SpringConfiguration {
}
```

```
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;
```

```
    /**
     * 创建一个 QueryRunner对象
     * @param dataSource
     * @return
     */
    @Bean(name = "runner")
    public QueryRunner creatQueryRunner(DataSource
dataSource){
        return  new QueryRunner(dataSource);
    }


    /**
     * 创建数据源，并且存入spring
     * @return
     */
    @Bean(name = "dataSource")
    public DataSource createDataSource() {
        try {
            ComboPooledDataSource ds = new
ComboPooledDataSource();
            ds.setUser(username);
            ds.setPassword(password);
            ds.setDriverClass(driver);
            ds.setJdbcUrl(url);
            return ds;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

6.5.导入其他配置类

@Import

这样看来一个 JdbcConfig 就基本写好了，我们在其中配置了 QueryRunner 对象，以及数据源，这个时候，实际上我们原先的 bean.xml 就可以删掉了，但是我们虽然写好了 JdbcConfig 但是如何将两个配置文件联系起来呢？这也就是这个注解的作用

```
@Configuration
@ComponentScan("cn.ideal")
@Import(JdbcConfig.class)
@PropertySource("classpath:jdbcConfig.properties")
public class SpringConfiguration {
}
```

7.注解获取容器

修改获取容器的方式后，就可以进行测试了

```
private ApplicationContext ac = new
AnnotationConfigApplicationContext(SpringConfiguration.cla
ss);

private AccountService as = ac.getBean("accountService",
AccountService.class);
```

8.Spring单元测试改进

由于我们需要通过上面测试中两行代获取到容器，为了不每一次都写这两行代码，所以我们在前面将其定义在了成员位置，但是有没有办法可以省掉这个步骤呢？

也就是说，我们想要程序自动创建容器，但是原来的 junit 很显然是实现不了的，因为它并不会知道我们是否使用了 spring ，不过 junit 提供了一个注解让我们替换它的运行器，转而由 spring 提供

首先需要导入 jar 包 或者说导入依赖坐标

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

使用 @RunWith 注解替换原有运行器 然后使用 @ContextConfiguration 指定 spring 配置文件的位置，然后使用 @Autowired 给变量注入数据

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = SpringConfiguration.class)
public class AccountServiceTest {

    @Autowired
    private AccountService as;
}
```