spring-day02

第一章 基于注解管理bean

第一节 IOC的相关注解及包扫描

1. 注解的优势

和 XML 配置文件一样，注解本身并不能执行，注解本身仅仅只是做一个标记，具体的功能是框架检测到注解标记的位置，然后针对这个位置按照注解标记的功能来执行具体操作。使用注解开发比使用XML更加简洁明了

2. 包扫描

使用注解进行Bean管理必须要在xml配置文件中进行包扫描，这样的话spring-framework才能够解析你在Bean上所添加的注解，包扫描包含下面列举的四种情况:

2.1 最基本的扫描方式[常用]

扫描指定包下的所有类以及其子包下的所有类

```
<!-- 配置自动扫描的包 -->
<!-- 最基本的扫描方式 -->
<context:component-scan base-package="com.atguigu.ioc.component"/>
```

2.2 指定匹配模式

根据具体的匹配规则，扫描某个包下的某些类，需要注意规则匹配是匹配的直接在这个包下的类，不能是子包下的类

```
<!-- 情况二：在指定扫描包的基础上指定匹配模式 -->
<context:component-scan
                        base-package="com.atguigu.ioc.component"
                        resource-pattern="Soldier*.class"/>
```

2.3 指定要排除的组件

扫描某个包下的所有类，但是排除掉一些注解

```
<!-- 情况三：指定不扫描的组件 -->
<context:component-scan base-package="com.atguigu.ioc.component">

    <!-- context:exclude-filter标签：指定排除规则 -->
    <!-- type属性：指定根据什么来进行排除，annotation取值表示根据注解来排除 -->
    <!-- expression属性：指定排除规则的表达式，对于注解来说指定全类名即可 -->
    <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
</context:component-scan>
```

2.4 仅扫描指定组件

扫描某个包下的所有类，但是只扫描某种注解

```
<!-- 情况四：仅扫描指定的组件 -->
<!-- 仅扫描 = 关闭默认规则 + 追加规则 -->
<!-- use-default-filters属性：取值false表示关闭默认扫描规则 -->
<context:component-scan base-package="com.atguigu.ioc.component" use-default-
filters="false">

    <!-- context:include-filter标签：指定在原有扫描规则的基础上追加的规则 -->
    <context:include-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
</context:component-scan>
```

3. 常用的进行IOC的注解

我们在类上添加注解，可以实现将该类的对象配置到spring的IOC容器中，常用的注解有如下四种:

3.1 Component注解

该注解主要用在**普通类**上，即除了三层结构之外的其它类的对象如果需要配置到spring的IOC容器中，那么则需要在类上添加Component注解

```
package com.atguigu.ioc.component;
import org.springframework.stereotype.Component;
@Component
public class CommonComponent {
}
```

3.2 Controller注解

该注解主要用在**控制层**的类上，控制器处于三层结构中的表现层，在JavaWeb阶段表现层使用的是Servlet，而在学习了spring-framework之后，表现层我们使用Controller代替

```
package com.atguigu.ioc.component;
import org.springframework.stereotype.Controller;
@Controller
public class SoldierController {
}
```

3.3 Service注解

该注解主要用在三层结构中的业务层的实现类上，用于将业务层的对象配置到spring的IOC容器中

```
package com.atguigu.ioc.component;
import org.springframework.stereotype.Service;
@Service
public class SoldierService implements ISoldierService{
}
```

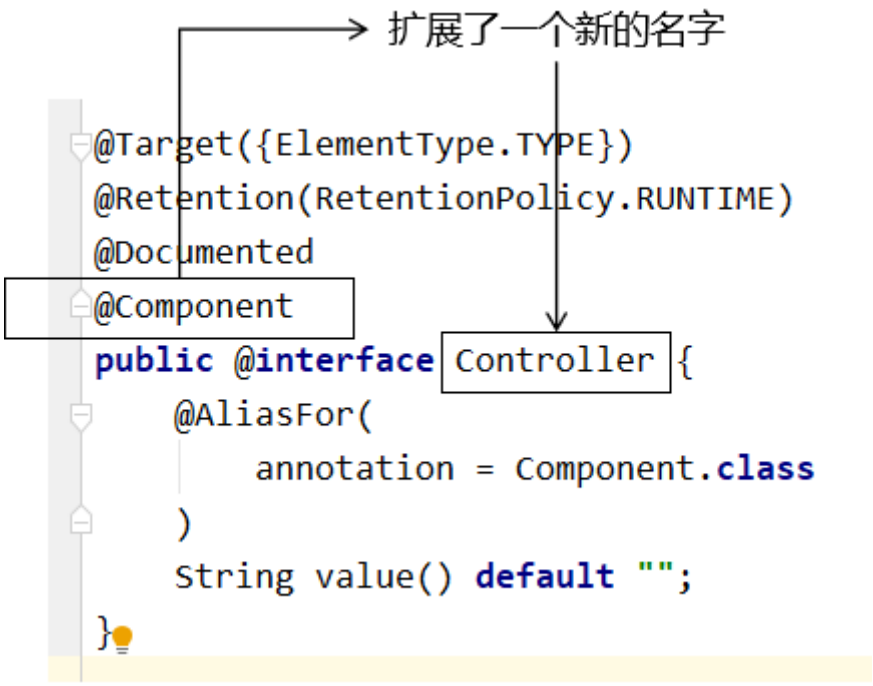
3.4 Repository注解

该注解主要用在三层结构中的持久层的实现类上，用于将持久层的对象配置到spring的IOC容器中，但是以后我们整合了Mybatis，这里就变成了Mapper接口，而**Mapper接口是由Mybatis和Spring的整合包负责扫描**的。由于Mybatis整合包想要把Mapper接口背后的代理类加入Spring的IOC容器需要结合Mybatis对Mapper配置文件的解析，所以这个事情是Mybatis和Spring的整合包来完成，将来由Mybatis负责扫描，也不需要使用Repository注解。

```
package com.atguigu.ioc.component;
import org.springframework.stereotype.Repository;
@Repository
public class SoldierDao implements ISoldierDao{
}
```

虽然我们学习的上述四个注解都实现IOC，但是其实他们四个在本质上是没有区别的。通过查看源码我们得知，@Controller、@Service、@Repository这三个注解只是在@Component注解的基础上起了三个新的名字。对于Spring使用IOC容器管理这些组件来说没有区别。所以@Controller、@Service、@Repository这三个注解只是给开发人员看的，让我们能够便于分辨组件的作用。

注意：虽然它们本质上一样，但是为了代码的可读性，为了程序结构严谨我们肯定不能随便胡乱标记。



4. 给Bean设置name

在我们使用XML方式管理bean的时候，**每个bean都有一个唯一标识，便于在其他地方引用。**现在使用注解后，**每个组件仍然应该有一个唯一标识。**

4.1 默认情况

类名首字母小写就是bean的id。例如：SoldierController类对应的bean的id就是soldierController。

4.2 使用value属性指定

```
@Controller(value = "tianDog")
public class SoldierController {
}
```

当注解中只设置一个属性时，value属性的属性名可以省略：

```
@Controller("tianDog")
public class SoldierController {
}
```

5. 怎么选择

1. 如果这个类是你自己写的类，表示你可以在它上面加注解，所以可以使用注解方式进行IOC
2. 如果这个类不是你自己写的类，而是第三方依赖中的类，表示你不能在它上面加注解，那么就只能用配置文件方式进行IOC

第二节 依赖注入相关的注解

2.1 注入简单类型的属性

Value注解是用于给IOC容器中的Bean注入简单类型的属性值

```
@Service("smallDog")
public class SoldierService {
    @Value("aobama")
    private String name;

}
```

2.2 读取properties中的数据并且进行注入

2.2.1 准备properties配置文件

happyInfo.properties

```
happy.componentName=aoalfu
```

2.2.2 使用PropertySource注解读取配置文件数据

```
package com.atguigu.component;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;

/**
 * 包名:com.atguigu.component
 *
 * @author Leevi
 * 日期2021-08-31  09:28
 * 当注解中只有一个属性需要赋值，并且这个属性的名字叫"value"的时候，可以省略"value = "
 */
@Component("happyComponent")
@PropertySource("classpath:happyInfo.properties")
public class HappyComponent {
    @Value("${happy.componentName}")
    private String componentName;

    public String getComponentName() {
        return componentName;
    }
```

```
}
```

2.3 注入Bean类型属性

2.3.1 设定情景

- UserController需要UserService
- UserService需要UserDao
- UserDao需要HappyComponent
- HappyComponent需要componentName

同时在各个组件中声明要调用的方法。

2.3.2 在各个组件中声明成员变量和方法

2.3.2.1 UserController

```
package com.atguigu.controller;

import com.atguigu.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

/**
 * 包名:com.atguigu.controller
 *
 * @author Leevi
 * 日期2021-08-29  16:50
 * 为了规范代码，提高代码的可读性，三层结构中不同层次的类使用不同的IOC注解
 * 1. Controller注解使用在表现层
 * 2. Service注解使用在业务层
 * 3. Repository注解使用在持久层
 * 4. Component注解使用在三层结构之外的类上
 *
 * 依赖注入的注解:
 * 1. Autowired注解: 表示自动装配,它只能用于注入Bean类型的对象
 * 2. Value注解: 注入简单类型的值
 */
@Controller("userController")
public class UserController {
    private UserService userService;
    public void printName(){
        System.out.println("打印:" + userService.getName());
    }
}
```

2.3.2.2 UserServiceImpl

```
package com.atguigu.service.impl;

import com.atguigu.dao.UserDao;
import com.atguigu.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * 包名:com.atguigu.service.impl
```

```
 *
 * @author Leevi
 * 日期2021-08-29  16:51
 */
@Service("userService")
public class UserServiceImpl implements UserService {
    private UserDao userDao;
    @Override
    public String getName() {
        return userDao.getName();
    }
}
```

2.3.2.3 UserDaoImpl

```
package com.atguigu.dao.impl;

import com.atguigu.component.HappyComponent;
import com.atguigu.dao.UserDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

/**
 * 包名:com.atguigu.dao.impl
 *
 * @author Leevi
 * 日期2021-08-29  16:52
 */
@Repository("userDao")
public class UserDaoImpl implements UserDao {
    private HappyComponent happyComponent;
    @Override
    public String getName() {
        //按道理来说要执行SQL语句到数据查询数据
        //但是今天通过依赖注入，模拟从数据库查询到name
        return happyComponent.getComponentName();
    }
}
```

2.3.2.4 HappyComponent

```
package com.atguigu.component;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * 包名:com.atguigu.component
 *
 * @author Leevi
 * 日期2021-08-31  09:28
 * 当注解中只有一个属性需要赋值，并且这个属性的名字叫"value"的时候，可以省略"value = "
 */
@Component("happyComponent")
public class HappyComponent {
    private String componentName;
```

```
    public String getComponentName() {
        return componentName;
    }
}
```

2.3.3 使用Autowired注解进行注入

2.3.3.1 Autowired注解可以使用的位置

1. 用在成员变量上(最常用)

```
package com.atguigu.controller;
import org.springframework.stereotype.Controller;
@Controller
public class UserController {
    @Autowired
    private UserService userService;
    public void printName(){
        System.out.println("打印:" + userService.getName());
    }
}
```

2. 用在构造器上

```
@Controller
public class UserController {
    private UserService userService;
    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }
}
```

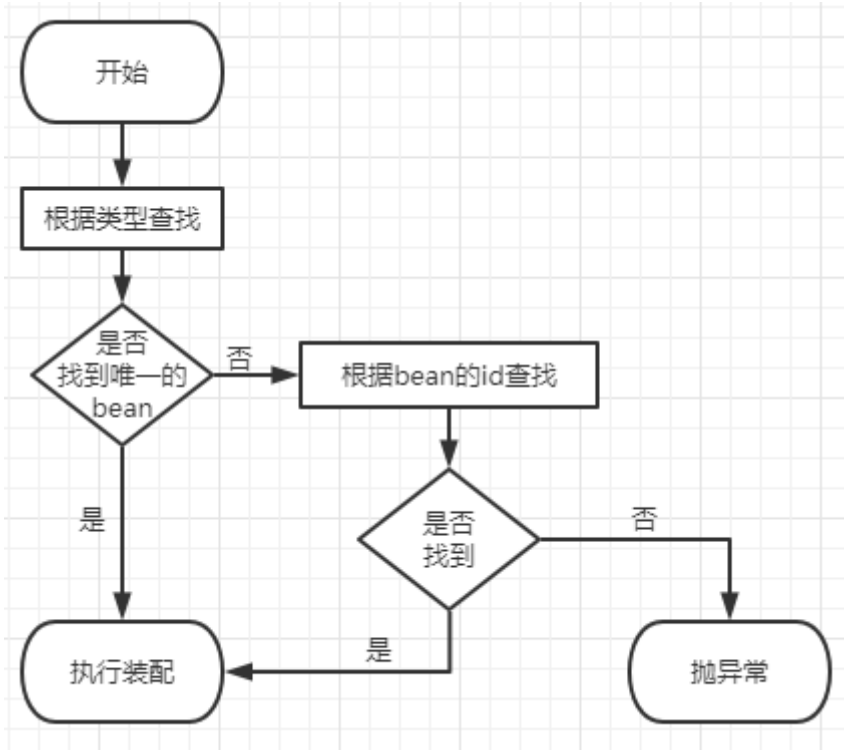
3. 用在方法上

```
@Controller
public class UserController {
    private UserService userService;
    @Autowired
    public void setUserService((UserService userService) {
        this.userService = userService;
    }
}
```

4. 用在参数上

```
@Controller
public class UserController {
    private UserService userService;
    public UserController(@Autowired UserService userService) {
        this.userService = userService;
    }
}
```

2.3.3.2 Autowired工作流程



首先根据所需要的组件类型到IOC容器中查找

- 能够找到唯一的bean：直接执行装配
- 如果完全找不到匹配这个类型的bean：装配失败
- 和所需类型匹配的bean不止一个
  - 没有@Qualifier注解：根据@Autowired标记位置成员变量的变量名作为bean的id进行匹配
    - 能够找到：执行装配
    - 找不到：装配失败
  - 使用@Qualifier注解：根据@Qualifier注解中指定的名称作为bean的id进行匹配
    - 能够找到：执行装配
    - 找不到：装配失败

2.3.4 使用Resource注解进行注入

```
package com.atguigu.ioc.component;
import org.springframework.stereotype.Controller;
@Controller
public class SoldierController {
    @Resource
    private SoldierService soldierService;
    public void getMessage() {
        soldierService.getMessage();
    }
}
```

2.3.5 Autowired注解和Resource注解的区别

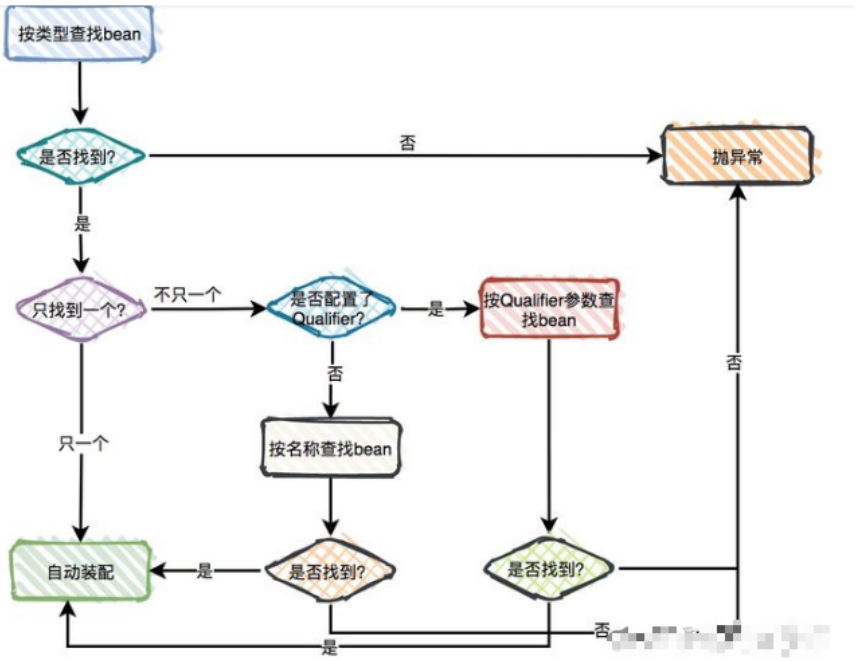
@Autowired功能虽说非常强大，但是也有些不足之处。比如：比如它跟spring强耦合了，如果换成了JFinal等其他框架，功能就会失效。而@Resource是JSR-250提供的，它是Java标准，绝大部分框架都支持。

除此之外，有些场景使用@Autowired无法满足的要求，改成@Resource却能解决问题。接下来，我们重点看看@Autowired和@Resource的区别。

- @**Autowired默认按byType**自动装配，而@Resource默认byName自动装配。
- @Autowired只包含**一个**参数：required，表示是否开启自动注入，默认是true。而@Resource包含**七个**参数，其中最重要的两个参数是：name 和 type。
- @Autowired如果要使用byName，需要使用@Qualifier一起配合。而@Resource如果指定了name，则用byName自动装配，如果指定了type，则用byType自动装配。
- @Autowired能够用在：构造器、方法、参数、成员变量和注解上，而@Resource能用在：类、成员变量和方法上。
- @Autowired是spring定义的注解，而@Resource是JSR-250定义的注解。

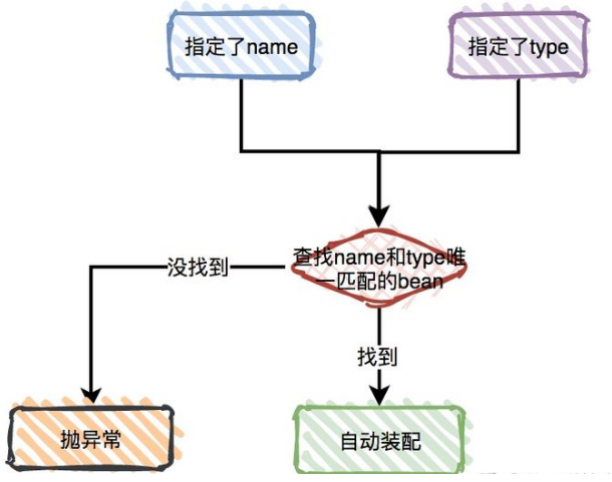
此外，它们的装配顺序不同。

Autowired的装配顺序:

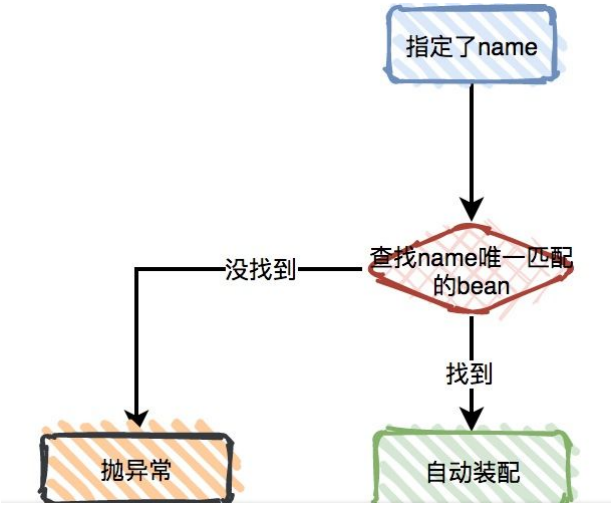


Resource的装配顺序:

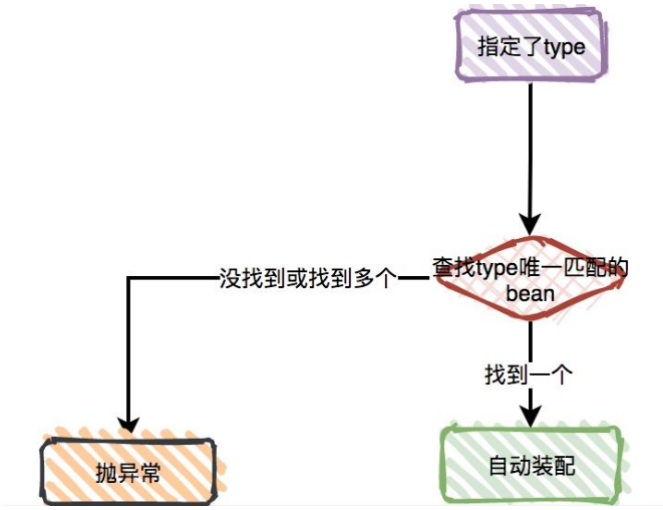
1. 如果同时指定了name和type：



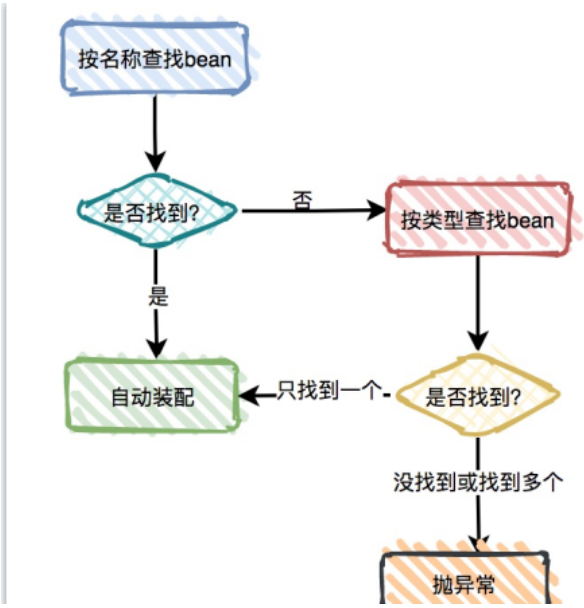
2. 如果指定了name：



3. 如果指定了type:



4. 如果既没有指定name，也没有指定type：



第三节 纯注解开发

体验纯注解开发，是为了给将来学习SpringBoot打基础。因为在SpringBoot中，就是完全舍弃XML配置文件，全面使用注解来完成主要的配置。

1. 使用配置类取代配置文件

1.1 创建配置类

使用@Configuration注解将一个普通的类标记为Spring的配置类。

```
package com.atguigu.configuration;

import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

import javax.sql.DataSource;

/**
 * 包名:com.atguigu.configuration
 *
 * @author Leevi
 * 日期2021-08-31  14:13
 * 1. 配置类上要添加一个@Configuration注解
 */
@Configuration
public class AtguiguSpringConfiguration {

}
```

1.2 在配置类中配置包扫描

```
package com.atguigu.configuration;

import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

import javax.sql.DataSource;

/**
 * 包名:com.atguigu.configuration
 *
 * @author Leevi
 * 日期2021-08-31  14:13
 * 1. 配置类上要添加一个@Configuration注解
 * 2. 使用@ComponentScan注解进行包扫描
 */
@Configuration
@ComponentScan("com.atguigu")
public class AtguiguSpringConfiguration {
}
```

1.3 在配置类中配置bean

对Bean进行IOC的时候，如果是自己编写的类，则可以直接通过IOC注解进行配置，如果是**非自己写的类**:例如JDK中或者第三方框架中的类，我们可以通过配置文件进行IOC；但是在纯注解中没有了配置文件，所以我们需要使用@Bean注解进行IOC

```
package com.atguigu.configuration;

import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

import javax.sql.DataSource;

/**
 * 包名:com.atguigu.configuration
 *
 * @author Leevi
 * 日期2021-08-31  14:13
 * 1. 配置类上要添加一个@Configuration注解
 * 2. 使用@ComponentScan注解进行包扫描
 * 3. 使用@Bean注解配置第三方的类的IOC
 * 4. 从properties文件中读取数据
 */
@Configuration
@ComponentScan("com.atguigu")
@PropertySource("classpath:jdbcInfo.properties")
public class AtguiguSpringConfiguration {
    @Value("${jdbc.username}")
    private String username;

    @Value("${jdbc.password}")
    private String password;

    @Value("${jdbc.url}")
    private String url;

    @Value("${jdbc.driver}")
    private String driver;
    @Bean
    public DataSource getDataSource(){
        DruidDataSource dataSource = new DruidDataSource();
        dataSource.setDriverClassName(driver);
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        return dataSource;
    }
}
```

1.4 根据配置类创建IOC容器对象

```
// AnnotationConfigApplicationContext根据配置类创建IOC容器对象
ApplicationContext iocContainerAnnotation = new
AnnotationConfigApplicationContext(MyConfiguration.class);
```

1.5 测试

```
package com.atguigu;

import com.atguigu.configuration.AtguiguSpringConfiguration;
import com.atguigu.controller.UserController;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

/**
 * 包名:com.atguigu
 *
 * @author Leevi
 * 日期2021-08-31  14:05
 */
public class TestAnnotationOnly {
    @Test
    public void testPrintConnection(){
        //1. 创建核心容器对象:AnnotationConfigApplicationContext核心容器是加载配置类的
核心容器
        ApplicationContext act = new
AnnotationConfigApplicationContext(AtguiguSpringConfiguration.class);
        //2. 从核心容器对象中获取UserController对象
        UserController userController = (UserController)
act.getBean("userController");
        //3. 调用UserController对象printConnection()方法
        userController.printConnection();
    }
}
```

第四节 Spring整合junit4

1. Spring整合Junit4的好处

- 好处1：不需要自己创建IOC容器对象了
- 好处2：任何需要的bean都可以在测试类中直接享受自动装配

2. 具体操作

2.1 加入依赖

```
<!--
            引入Spring整合Junit的依赖
        -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.3.1</version>
    <scope>test</scope>
</dependency>
```

2.2 创建测试类

如果是使用的配置类:

```
package com.atguigu;

import com.atguigu.configuration.AtguiguSpringConfiguration;
import com.atguigu.controller.UserController;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * 包名:com.atguigu
 *
 * @author Leevi
 * 日期2021-08-31  14:05
 * 目标:在单元测试类中注入要测试的对象,这样就不需要自己创建核心容器了，也不需要自己调用
getBean()方法从核心容器中获取对象了
 *
 * 实现方案:使用Spring整合Junit单元测试
 * 实现步骤:
 * 1. 引入spring整合Junit的依赖
 * 2. 让单元测试类依赖SpringJUnit4ClassRunner来运行:在测试类上添加
@RunWith(SpringJUnit4ClassRunner.class)
 * 3. 加载配置文件或者配置类: 在测试类上添加@ContextConfiguration(classes =
AtguiguSpringConfiguration.class)
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = AtguiguSpringConfiguration.class)
public class TestAnnotationOnly {
    @Autowired
    private UserController userController;
    @Test
    public void testPrintConnection(){
        //3. 调用UserController对象printConnection()方法
        userController.printConnection();
    }
}
```

如果是使用的配置文件:

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:spring-application.xml")
public class TestSpringAnnotation {
    @Autowired
    private UserController userController;
    @Test
    public void testPrintConnection(){
        //3. 调用UserController对象printConnection()方法
        userController.printConnection();
    }
}
```

第五节 内容总结

1. 包扫描: 指定要进行组件扫描的包
    1. 基础扫描(用的最多)
    2. 指定模式的扫描
    3. 排除某些注解
    4. 只扫描某些注解
2. IOC 注解:
    1. Component注解: 三层结构之外的其他类使用
    2. Controller注解: 表现层的类使用
    3. Service注解: 业务层的类使用
    4. Repository注解: 持久层的类使用
3. 依赖注入注解:
    1. 注入Bean:
        1. Autowired注解，它是byType进行自动注入，如果要byName必须结合Qualifier注解一起使用
        2. Resource注解，它默认是byName进行自动注入
    2. 注入简单类型: Value注解，以及使用PropertySource注解引入外部的properties文件
4. 注解方式和配置文件方式进行IOC和依赖注入的选择问题:
    1. 如果是自己写的类就使用注解方式
    2. 如果是第三方jar中的类就使用配置文件方式
5. Spring整合Junit:
    1. 目的：简化单元测试
    2. 实现步骤:
        1. 引入spring-test的依赖
        2. Junit的依赖版本必须是4.12及以上
        3. 给单元测试类添加@Runwith(SpringJUnit4ClassRunner.class)
        4. 给单元测试类添加@ContextConfiguration(locations="配置文件的路径"或者是classes=配置类.class)
        5. 直接注入你想使用的IOC容器中的对象就可以直接使用了
6. Spring的纯注解开发
    1. 目的: 为了以后学习SpringBoot做准备，我们项目如果是使用Spring做开发的话是不会用纯注解的
    2. 步骤:
        1. 配置类上要添加@Configuration注解标示为配置类
        2. 配置类上要添加@ComponentScan指定要扫描的包

3. 如果要对第三方的类进行IOC配置
    1. 在配置类中创建一个方法
        1. 修饰符public
        2. 返回值是要进行IOC的对象的类型
        3. 方法体中编写创建IOC对象的代码
        4. 如果要给这个方法注入一个IOC容器中存在的对象，直接在方法的参数中声明就行了
    2. 给该方法添加@Bean注解
4. 如果是使用纯注解开发，整合Junit的时候，@ContextConfiguration(classes=配置类.class)