Spring 第二天

第1章 基于注解的 IOC 配置

1.1 写在最前

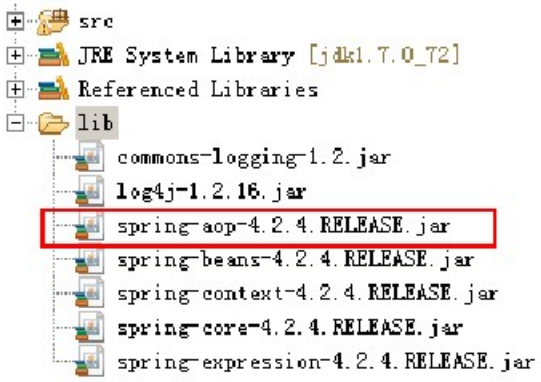
学习基于注解的 IoC 配置，大家脑海里首先得有一个认知，即注解配置和 xml 配置要实现的功能都是一样的，都是要降低程序间的耦合。只是配置的形式不一样。

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。

1.2 环境搭建

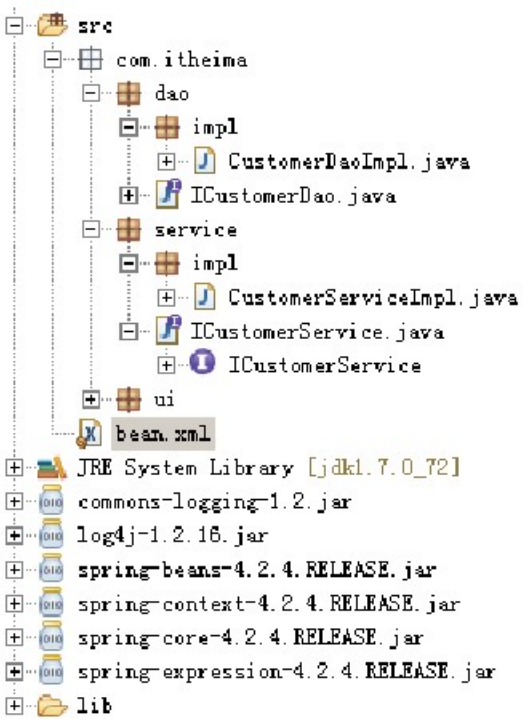
1.2.1 第一步：拷贝必备 jar 包到工程的 lib 目录。

注意：在基于注解的配置中，我们还要多拷贝一个 aop 的 jar 包。如下图：



1.2.2 第二步：在类的根路径下创建一个任意名称的 xml 文件（不能是中文）

基于注解整合时，导入约束时需要多导入一个 context 名称空间下的约束。



给配置文件导入约束：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 导入 schema
约束的位置在：
    ..\spring-framework-4.2.4.RELEASE\docs\spring-framework-reference\html\xsd-configuration.html
    文件中。
注意：要导入 schema 约束
-->
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      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 ">
</beans>
```

1.2.3 第三步：使用@Component 注解配置管理的资源

```
/**
 * 客户的业务层实现类
 * @author zhy
 *
```

```
@Component(value="customerService")
public class CustomerServiceImpl implements ICustomerService {
    @Override
    public void saveCustomer() {
        System.out.println("执行了保存客户");
    }
}
```

1.2.4 第四步在 spring 的配置文件中开启 spring 对注解 ioc 的支持

```
<!-- 告知 spring 框架在，读取配置文件，创建容器时，扫描注解，依据注解创建对象，并存入容器中 -->
    <context:component-scan base-package="com.itheima"></context:component-scan>
```

1.3 常用注解

1.3.1 用于创建对象的

相当于：<bean id="" class="">

1.3.1.1 @Component

作用：

把资源让 spring 来管理。相当于在 xml 中配置一个 bean。

属性：

value：指定 bean 的 id。如果不指定 value 属性，默认 bean 的 id 是当前类的类名。首字母小写。

1.3.1.2 @Controller @Service @Repository

他们三个注解都是针对一个的衍生注解，他们的作用及属性都是一模一样的。

他们只不过是提供了更加明确的语义化。

@Controller：一般用于表现层的注解。

@Service：一般用于业务层的注解。

@Repository：一般用于持久层的注解。

细节：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。

1.3.2 用于注入数据的

相当于：<property name="" ref="">

<property name="" value="">

1.3.2.1 @Autowired

作用：

自动按照类型注入。当使用注解注入属性时，set 方法可以省略。它只能注入其他 bean 类型。当有多个类型匹配时，使用要注入的对象变量名称作为 bean 的 id，在 spring 容器查找，找到了也可以注入成功。找不到就报错。

1.3.2.2 @Qualifier

作用：

在自动按照类型注入的基础之上，再按照 Bean 的 id 注入。它在给字段注入时不能独立使用，必须和@Autowire 一起使用；但是给方法参数注入时，可以独立使用。

属性：

value：指定 bean 的 id。

1.3.2.3 @Resource

作用：

直接按照 Bean 的 id 注入。它也只能注入其他 bean 类型。

属性：

name：指定 bean 的 id。

1.3.2.4 @Value

作用：

注入基本数据类型和 String 类型数据的

属性：

value：用于指定值

1.3.3 用于改变作用范围的：

相当于：<bean id="" class="" scope="">

1.3.3.1 @Scope

作用：
　　指定bean的作用范围。
属性：
　　value：指定范围的值。
　　　　取值：singleton　prototype request session globalsession

1.3.4 和生命周期相关的：(了解)

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

1.3.4.1 @PostConstruct

作用：
　　用于指定初始化方法。

1.3.4.2 @PreDestroy

作用：
　　用于指定销毁方法。

1.3.5 代码示例

业务层代码：

```
/**
 * 客户的业务层接口
 */
public interface ICustomerService {
    /**
     * 保存客户
     * @param customer
     */
    void saveCustomer();
}


/**
 * 客户的业务层实现类
 */
```

```
//作用就相当于在 xml 中配置了一个 bean 标签，该注解有 value 属性，含义是 bean 的 id。
//不写的时候，默认的 id 是：当前类名，且首字母小写。即：customerServiceImpl
@Component(value="customerService")
@Scope(value="singleton")
public class CustomerServiceImpl implements ICustomerService {
//  @Autowired
//  自动按照数据类型注入，拿着当前变量的数据类型在 spring 的容器中找，找到后，给变量赋值。
//  当有多个类型匹配时，会使用当前变量名称 customerDao 作为 bean 的 id，继续在容器中找。
//  找到了，也能注入成功。找不到就报错。
//  @Qualifier(value="customerDao2")//在自动按照类型注入的基础之上，再按照 id 注入
    @Resource(name="customerDao2")//直接按照 bean 的 id 注入
    private ICustomerDao customerDao = null;

    @Value("com.mysql.jdbc.Driver")//注入基本类型和 String 类型数据
    private String driver;

    @Override
    public void saveCustomer() {
        System.out.println(driver);
        customerDao.saveCustomer();
    }
}
```

持久层代码：

```
/**
 * 客户的持久层接口
 */
public interface ICustomerDao {
    /**
     * 保存客户
     */
    void saveCustomer();
}
```

```
/**
 * 客户的持久层实现类 1111111111111111111
 */
@Repository("customerDao1")
public class CustomerDaoImpl implements ICustomerDao {
```

```
        @Override
        public void saveCustomer() {
            System.out.println("保存了客户 1111111111111111");
        }
    }

    /**
     * 客户的持久层实现类 22222222222222222222
     */
    @Repository("customerDao2")
    public class CustomerDaoImpl2 implements ICustomerDao {

        @Override
        public void saveCustomer() {
            System.out.println("保存了客户 2222222222222222");
        }
    }
```

测试类代码：

```
    public class Client {
        public static void main(String[] args) {
            //1.获取容器
            ApplicationContext ac = new
ClassPathXmlApplicationContext("bean.xml");
            //2.根据 id 获取对象
            ICustomerService cs = (ICustomerService)
ac.getBean("customerService");
            cs.saveCustomer();
        }
    }
```

配置文件：

```
    <?xml version="1.0" encoding="UTF-8"?>
    <!-- 我们导入约束时，除了昨天的那部分之外，还要单独导入一个 context 名称空间 -->
    <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"
           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">
        <!-- 告知 spring 框架在通过读取配置文件创建容器时，扫描的包，并根据包中类的注解创
```

```
建对象-->
        <context:component-scan
base-package="com.itheima"></context:component-scan>
    </beans>
```

1.3.6 关于 Spring 注解和 XML 的选择问题

注解的优势：

配置简单，维护方便（我们找到类，就相当于找到了对应的配置）。

XML 的优势：

修改时，不用改源码。不涉及重新编译和部署。

Spring 管理 **Bean** 方式的比较：

	基于XML配置	基于注解配置
Bean定义	<bean id="..." class="..." />	@Component 衍生类@Repository @Service @Controller
Bean名称	通过 id或name 指定	@Component("person")
Bean注入	<property> 或者通过p命名空间	@Autowired 按类型注入 @Qualifier按名称注入
生命过程、Bean作用范围	init-method destroy-method 范围 scope属性	@PostConstruct 初始化 @PreDestroy 销毁 @Scope设置作用范围
适合场景	Bean来自第三方，使用其它	Bean的实现类由用户自己开发

1.4spring 管理对象细节

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。

写到此处，基于注解的 IoC 配置已经完成，但是大家都发现了一个问题：我们依然离不开 spring 的 xml 配置文件，那么能不能不写这个 bean.xml，所有配置都用注解来实现呢？

答案是肯定的，请看下一章节。

1.5spring 的纯注解配置

1.5.1 待改造的问题

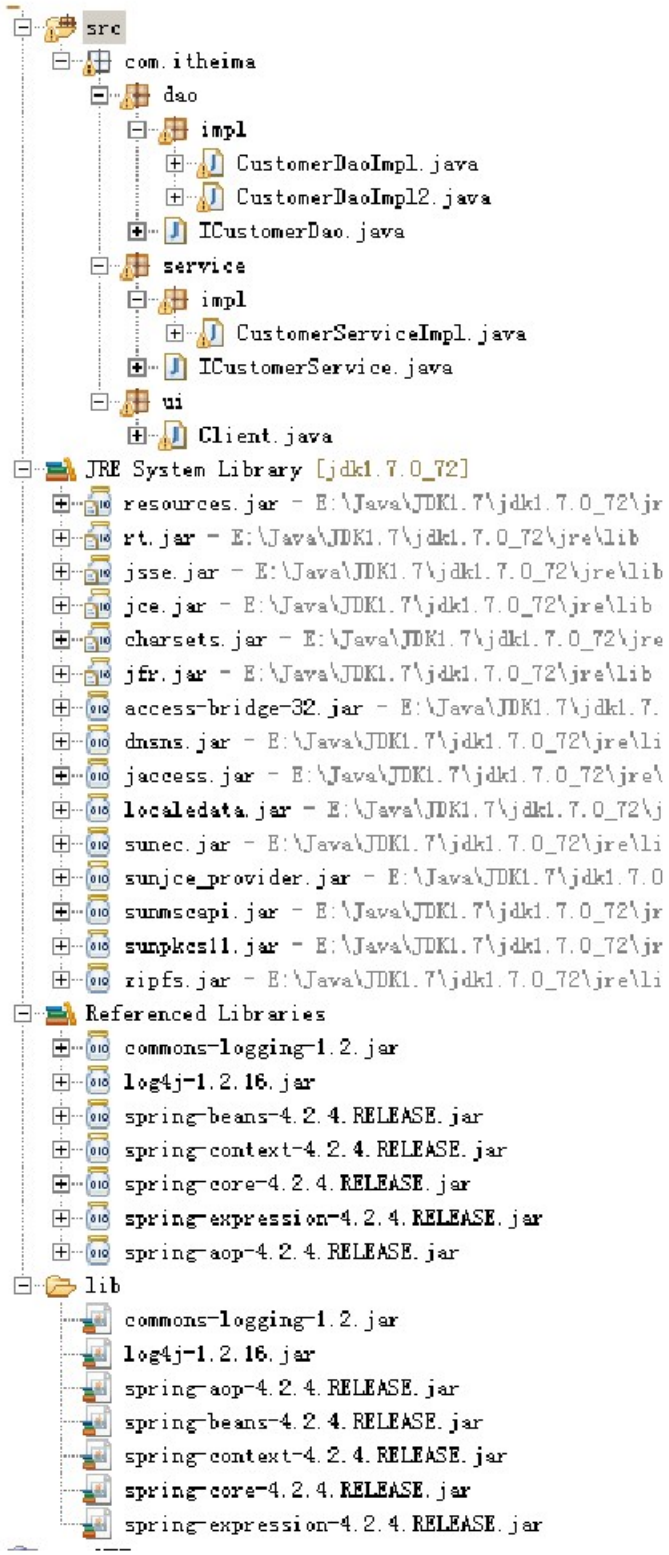
我们发现，之所以我们现在离不开 xml 配置文件，是因为我们有一句很关键的配置：

```
    <!-- 告知 spring 框架在，读取配置文件，创建容器时，扫描注解，依据注解创建对象，并存入容器中 -->
        <context:component-scan base-package="com.itheima"></context:component-scan>
```

如果他要也能用注解配置，那么我们就可以脱离 xml 文件了。

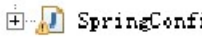
1.5.2 使用注解配置要扫描的包

工程结构，如下图：



在此图中，我们已经一点也看不到 xml 的身影了。那么，那句关键的配置跑哪去了呢？

在一个新的类上：




```
/**
 * 客户的业务层实现类
 */
```

```
@Configuration//表明当前类是一个配置类
@ComponentScan(basePackages = "com.itheima")//配置要扫描的包
public class SpringConfiguration {
}
```

那么新的问题又来了，我们如何获取容器呢？

```
public class Client {
    public static void main(String[] args) {
        //1.获取容器：由于我们已经没有了 xml 文件，所以再用读取 xml 方式就不能用了。
        //这时需要指定加载哪个类上的注解
        ApplicationContext ac =
            new
AnnotationConfigApplicationContext(SpringConfiguration.class);
        //2.根据 id 获取对象
        ICustomerService          cs          =          (ICustomerService)
ac.getBean("customerService");
        cs.saveCustomer();
    }
}
```

1.5.3 新注解说明

1.5.3.1 @Configuration

作用：

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)。

属性：

value:用于指定配置类的字节码

示例代码：

```
/**
 * 用于初始化 spring 容器的配置类
 */
@Configuration
public class SpringConfiguration{
}
```

1.5.3.2 @ComponentScan

作用：

用于指定 spring 在初始化容器时要扫描的包。作用和在 spring 的 xml 配置文件中的：

<context:component-scan base-package="com.itheima"/>是一样的。

属性：

basePackages：用于指定要扫描的包。和该注解中的 value 属性作用一样。

1.5.3.3 @PropertySource

作用：

用于加载.properties 文件中的配置。例如我们配置数据源时，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。

属性：

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

示例代码：

配置：

```
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;

    @Bean(name="dataSource")
    public DataSource getDataSource(){
        BasicDataSource ds = new BasicDataSource();
        ds.setDriverClassName(driver);
        ds.setUrl(url);
        ds.setUsername(username);
        ds.setPassword(password);
        return ds;
    }
}
```

jdbc.properties 文件：

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/day44_ee247_spring
jdbc.username=root
jdbc.password=1234
```

注意：

我们目前上课使用的版本是 **4.2.4**，在 **spring4.3** 以前都需要提供一个占位符配置器：

PropertySourcesPlaceholderConfigurer

而在 **spring4.3** 以后，则不需要提供。

提供的方式如下：（在 SpringConfiguration 或 JdbcConfig 中配置均可）

```
    @Bean
    public static PropertySourcesPlaceholderConfigurer
                                    propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }
```

1.5.3.4 @Import

作用：

用于导入其他配置类，在引入其他配置类时，可以不用再写@Configuration注解。当然，写上也没问题。

属性：

value[]：用于指定其他配置类的字节码。

示例代码：

```
@Configuration
@ComponentScan(basePackages = "cn.itcast.spring")
@Import({ Configuration_B.class})
public class Configuration_A {
}

@Configuration
@PropertySource("classpath:info.properties")
public class Configuration_B {

}
```

1.5.3.5 @Bean

作用：

该注解只能写在方法上，表明使用此方法创建一个对象，并且放入 spring 容器。它就相当于我们之前在 xml 配置中介绍的 factory-bean 和 factory-method。

属性：

name：给当前@Bean注解方法创建的对象指定一个名称(即 bean 的 id)。

示例代码：

```
@Bean(name = "datasource2")
    public DataSource createDS() throws Exception {
        ComboPooledDataSource comboPooledDataSource = new
ComboPooledDataSource();
        comboPooledDataSource.setUser("root");
        comboPooledDataSource.setPassword("1234");
        comboPooledDataSource.setDriverClass("com.mysql.jdbc.Driver");
        comboPooledDataSource.setJdbcUrl("jdbc:mysql:///spring_ioc");
        return comboPooledDataSource;
```

```
    }
```

第2章 Spring 整合 Junit

2.1 准备测试环境

2.1.1 创建业务层接口实现类

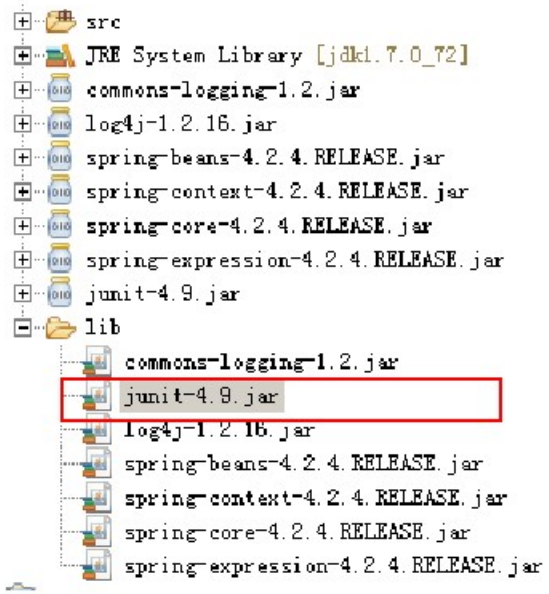
```
/**
 * 客户的业务层接口
 */
public interface ICustomerService {

    /**
     * 查询所有客户
     * @return
     */
    List<Customer> findAllCustomer();

    /**
     * 保存客户
     * @param customer
     */
    void saveCustomer(Customer customer);
}

/**
 * 客户的业务层实现类
 */
public class CustomerServiceImpl implements ICustomerService {

    private ICustomerDao customerDao;

    public void setCustomerDao(ICustomerDao customerDao) {
        this.customerDao = customerDao;
    }

    @Override
    public List<Customer> findAllCustomer() {
        return customerDao.findAllCustomer();
    }
```

```
    @Override
    public void saveCustomer(Customer customer) {
        customerDao.save(customer);
    }

}
```

2.1.2 创建持久层接口实现类

```
/**
 * 客户的持久层接口
 */
public interface ICustomerDao {

    /**
     * 查询所有客户
     * @return
     */
    List<Customer> findAllCustomer();

    /**
     * 保存客户
     * @param customer
     */
    void save(Customer customer);
}

/**
 * 客户的持久层实现类
 */
public class CustomerDaoImpl implements ICustomerDao {

    @Override
    public List<Customer> findAllCustomer() {
        System.out.println("查询了所有客户");
        return null;
    }

    @Override
    public void save(Customer customer) {
        System.out.println("保存了客户");
    }
}
```

2.1.3 导入 junit 的 jar 包



2.1.4 编写测试类

```
/**
 * 测试客户的业务层和持久层
 */
public class CustomerServiceTest {

    private ICustomerService customerService;

    @Test
    public void testFindAll(){
        customerService.findAllCustomer();
    }

    @Test
    public void testSave(){
        Customer c = new Customer();
        c.setCustName("传智学院 ");
        customerService.saveCustomer(c);
    }
}
```

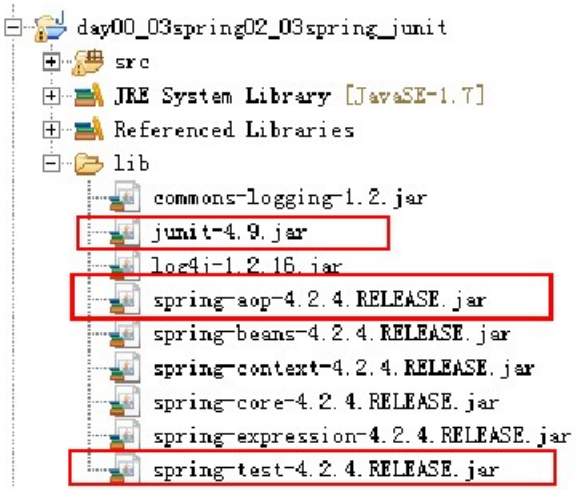
2.2 使用 xml 配置步骤

2.2.1 xml 文件中的配置

```
<?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">
       <!-- 把资源交给 spring 来管理 -->
       <bean                                                id="customerDao"
class="com.itheima.dao.impl.CustomerDaoImpl"></bean>

       <bean                                              id="customerService"
class="com.itheima.service.impl.CustomerServiceImpl">
           <property name="customerDao" ref="customerDao"></property>
       </bean>
</beans>
```

2.2.2 第一步：拷贝整合 junit 的必备 jar 包到 lib 目录

此处需要注意的是，导入 jar 包时，需要导入一个 spring 中 aop 的 jar 包。



2.2.3 第二步：使用@RunWith 注解替换原有运行器

```
@RunWith(SpringJUnit4ClassRunner.class)
public class CustomerServiceTest {
```

```
    private ICustomerService customerService;

    @Test
    public void testFindAll(){
        customerService.findAllCustomer();
    }

    @Test
    public void testSave(){
        Customer c = new Customer();
        c.setCustName("传智学院 ");
        customerService.saveCustomer(c);
    }
}
```

2.2.4 第三步：使用@ContextConfiguration 指定 spring 配置文件的位置

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:bean.xml"})
public class CustomerServiceTest {

    private ICustomerService customerService;

    @Test
    public void testFindAll(){
        customerService.findAllCustomer();
    }

    @Test
    public void testSave(){
        Customer c = new Customer();
        c.setCustName("传智学院 ");
        customerService.saveCustomer(c);
    }
}
```

2.2.5 第四步：使用@Autowired 给测试类中的变量注入数据

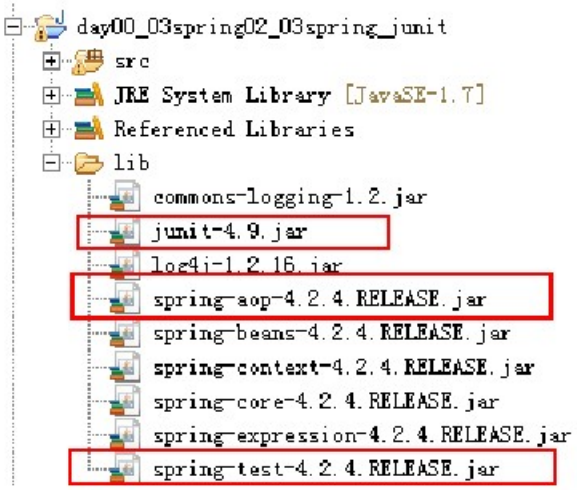
```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:bean.xml"})
```

```
public class CustomerServiceTest {

    @Autowired
    private ICustomerService customerService;

    @Test
    public void testFindAll(){
        customerService.findAllCustomer();
    }

    @Test
    public void testSave(){
        Customer c = new Customer();
        c.setCustName("传智学院 ");
        customerService.saveCustomer(c);
    }
}
```

2.3 使用纯注解配置步骤

2.3.1 第一步：拷贝整合 junit 的必备 jar 包到 lib 目录

此处需要注意的是，导入 jar 包时，需要导入一个 spring 中 aop 的 jar 包。



2.3.2 第二步：把资源都用注解管理

```
@Service("customerService")
public class CustomerServiceImpl implements ICustomerService {

    @Autowired
```

```
    private ICustomerDao customerDao;

    @Override
    public List<Customer> findAllCustomer() {
        return customerDao.findAllCustomer();
    }

    @Override
    public void saveCustomer(Customer customer) {
        customerDao.save(customer);
    }
}

/**
 * 客户的持久层实现类
 */
@Repository("customerDao")
public class CustomerDaoImpl implements ICustomerDao {

    @Override
    public List<Customer> findAllCustomer() {
        System.out.println("查询了所有客户");
        return null;
    }

    @Override
    public void save(Customer customer) {
        System.out.println("保存了客户");
    }
}
```

2.3.3 第三步：使用注解配置方式创建 spring 容器

```
@Configuration
@ComponentScan(basePackages={"com.itheima"})
public class CustomerServiceTest {

    @Autowired
    private ICustomerService customerService;

    @Test
    public void testFindAll(){
        customerService.findAllCustomer();
    }
```

```
    @Test
    public void testSave(){
        Customer c = new Customer();
        c.setCustName("传智学院 ");
        customerService.saveCustomer(c);
    }
}
```

2.3.4 第四步：使用 RunWith 注解和 ContextConfiguration 注解配置

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes={CustomerServiceTest.class})
@Configuration
@ComponentScan(basePackages={"com.itheima"})
public class CustomerServiceTest {

    @Autowired
    private ICustomerService customerService;

    @Test
    public void testFindAll(){
        customerService.findAllCustomer();
    }

    @Test
    public void testSave(){
        Customer c = new Customer();
        c.setCustName("传智学院 ");
        customerService.saveCustomer(c);
    }
}
```

2.4 为什么不把测试类配到 xml 中

在解释这个问题之前，先解除大家的疑虑，配到 XML 中能不能用呢？
答案是肯定的，没问题，可以使用。
那么为什么不采用配置到 xml 中的方式呢？
这个原因是这样的：

第一：当我们在 xml 中配置了一个 bean，spring 加载配置文件创建容器时，就会创建对象。

第二：测试类只是我们在测试功能时使用，而在项目中它并不参与程序逻辑，也不会解

决需求上的问题，所以创建完了，并没有使用。那么存在容器中就会造成资源的浪费。

所以，基于以上两点，我们不应该把测试配置到 xml 文件中。