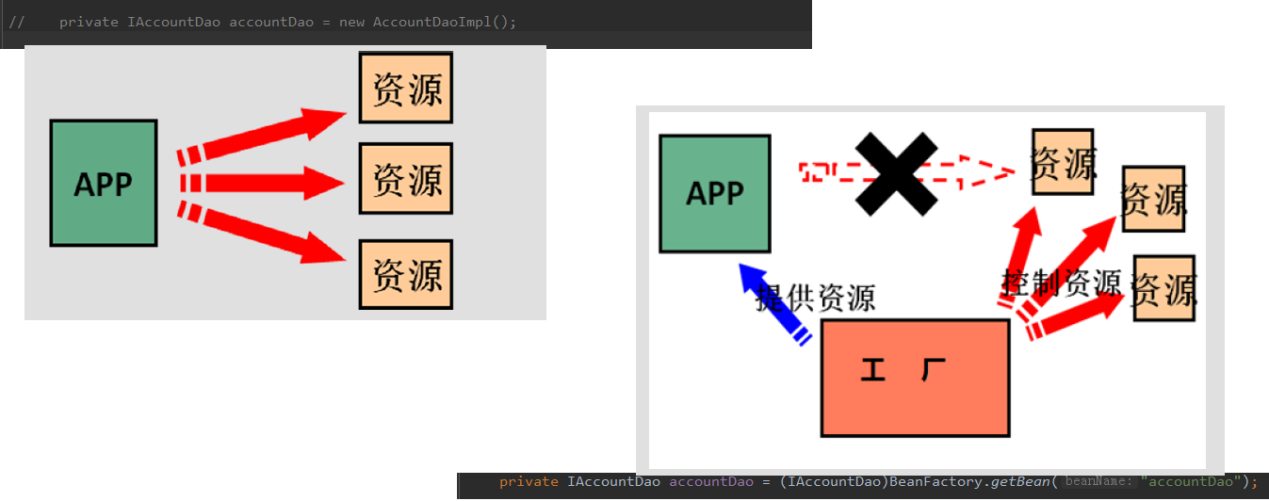
**Spring**

Spring 是分层的 Java SE/EE 应用 full-stack 轻量级开源框架，以 IoC（Inverse Of Control： 反转控制）和 AOP（Aspect Oriented Programming：面向切面编程）为内核。



**明确 ioc 的作用**：

削减计算机程序的耦合(解除我们代码中的依赖关系)。

**实例化 Bean 的三种方式:**

**第一种方式：使用默认无参构造函数**

<!--在默认情况下：

它会根据默认无参构造函数来创建类对象。如果 bean 中没有默认无参构造函数，将会创建失败。-->

<bean id=*"accountService"* class=*"com.itheima.service.impl.AccountServiceImpl"*/>

**第二种方式：spring 管理静态工厂-使用静态工厂的方法创建对象**

/\*\*

\* 模拟一个静态工厂，创建业务层实现类

\*/

**public class** StaticFactory {

**public static** IAccountService createAccountService(){

**return new** AccountServiceImpl();

}

}

<!-- 此种方式是:

使用 StaticFactory 类中的静态方法 createAccountService 创建对象，并存入 spring 容器

id 属性：指定 bean 的 id，用于从容器中获取

class 属性：指定静态工厂的全限定类名

factory-method 属性：指定生产对象的静态方法

-->

<bean id=*"accountService"*

class=*"com.itheima.factory.StaticFactory"*

factory-method=*"createAccountService"*></bean>

**第三种方式：spring 管理实例工厂-使用实例工厂的方法创建对象**

/\*\*

\* 模拟一个实例工厂，创建业务层实现类

\* 此工厂创建对象，必须现有工厂实例对象，再调用方法

\*/

**public class** InstanceFactory {

**public** IAccountService createAccountService(){

**return new** AccountServiceImpl();

}

}

<!-- 此种方式是：

先把工厂的创建交给 spring 来管理。

然后在使用工厂的 bean 来调用里面的方法

factory-bean 属性：用于指定实例工厂 bean 的 id。

factory-method 属性：用于指定实例工厂中创建对象的方法。

-->

<bean id=*"instancFactory"* class=*"com.itheima.factory.InstanceFactory"*></bean>

<bean id=*"accountService"*

factory-bean=*"instancFactory"*

factory-method=*"createAccountService"*></bean>

**spring 的依赖注入**

依赖注入：**Dependency Injection**。它是 spring 框架核心 ioc 的具体实现。

<!-- 通过配置文件给 bean 中的属性传值：使用 set 方法的方式涉及的标签：**property**

属性：

**name**：找的是类中 set 方法后面的部分

**ref**：给属性赋值是其他 bean 类型的

**value**：给属性赋值是基本数据类型和 string 类型的

实际开发中，此种方式用的较多。

-->

<bean id=*"accountService"* class=*"com.itheima.service.impl.AccountServiceImpl"*>

<property name=*"name"* value=*"test"*></property>

<property name=*"age"* value=*"21"*></property>

<property name=*"birthday"* ref=*"now"*></property>

</bean>

<bean id=*"now"* class=*"java.util.Date"*></bean>

<!-- 注入集合数据

List 结构的：

**array,list,set**

Map 结构的

**map,entry,props,prop**

-->

<bean id=*"accountService"* class=*"com.itheima.service.impl.AccountServiceImpl"*>

<!-- 在注入集合数据时，只要结构相同，标签可以互换 -->

<!-- 给数组注入数据 -->

<property name=*"myStrs"*>

<set>

<value>AAA</value>

<value>BBB</value>

<value>CCC</value>

</set>

</property>

<!-- 注入 list 集合数据 -->

<property name=*"myList"*>

<array>

<value>AAA</value>

<value>BBB</value>

<value>CCC</value>

</array>

</property>

<!-- 注入 set 集合数据 -->

<property name=*"mySet"*>

<list>

<value>AAA</value>

<value>BBB</value>

<value>CCC</value>

</list>

</property>

<!-- 注入 Map 数据 -->

<property name=*"myMap"*>

<props>

<prop key=*"testA"*>aaa</prop>

<prop key=*"testB"*>bbb</prop>

</props>

</property>

<!-- 注入 properties 数据 -->

<property name=*"myProps"*>

<map>

<entry key=*"testA"* value=*"aaa"*></entry>

<entry key=*"testB"*>

<value>bbb</value>

</entry>

</map>

</property>

</bean>

**配置步骤**

**配置对象**

<?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"*>

<property name=*"accountDao"* ref=*"accountDao"*></property>

</bean>

<!-- 配置 dao -->

<bean id=*"accountDao"* class=*"com.itheima.dao.impl.AccountDaoImpl"*>

<property name=*"dbAssit"* ref=*"dbAssit"*></property>

</bean>

<!-- 配置 dbAssit **此处我们只注入了数据源，表明每条语句独立事务**-->

<bean id=*"dbAssit"* class=*"com.itheima.dbassit.DBAssit"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</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:///spring\_day02"*></property>

<property name=*"user"* value=*"root"*></property>

<property name=*"password"* value=*"1234"*></property>

</bean>

</beans>

测试：

/\*\*\* 测试保存\*/

@Test

**public void** testSaveAccount() {

Account account = **new** Account();

account.setName("黑马程序员");

account.setMoney(100000f);

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

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

as.saveAccount(account);

}

**基于注解的 IOC 配置**

**1、先导入pom.xml坐标**

**2、业务层实现：**

@Service("accountService")

**public class** AccountServiceImpl **implements** IAccountService {

**private** IAccountDao accountDao;

**public void** setAccountDao(IAccountDao accountDao) {

**this**.accountDao = accountDao;

}

}

1. **创建 spring 的 xml 配置文件并开启对注解的支持：**

注意：

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。

由于我们使用了注解配置，此时不能在继承 JdbcDaoSupport，需要自己配置一个 JdbcTemplate

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<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"*>

<!-- 告知 spring 创建容器时要扫描的包 -->

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

<!-- 配置 dbAssit -->

<bean id=*"dbAssit"* class=*"com.itheima.dbassit.DBAssit"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</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:///spring\_day02"*></property>

<property name=*"user"* value=*"root"*></property>

<property name=*"password"* value=*"1234"*></property>

</bean>

</beans>

**常用注解：**

**@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。**

**@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。

**AOP 的相关概念**

AOP：全称是 Aspect Oriented Programming 即：面向切面编程。

简单的说它就是把我们程序重复的代码抽取出来，在需要执行的时候，使用动态代理的技术，在不修改源码的 基础上，对我们的已有方法进行增强。

**Joinpoint(连接点):**

所谓连接点是指那些被拦截到的点。在 spring 中,这些点指的是方法,因为 spring 只支持方法类型的

连接点。

**Pointcut(切入点):**

所谓切入点是指我们要对哪些 Joinpoint 进行拦截的定义。

关于转账的事务配置：

1. xml配置:

ServiceImpl:

**public class** AccountServiceImpl2 **implements** IAccountService{  
  
 **private** IAccountDao **accountDao**;  
  
 **public void** setAccountDao(IAccountDao accountDao) {  
 **this**.**accountDao** = accountDao;  
 }  
  
 *//Spring配置事务模板* **private** TransactionTemplate **transactionTemplate**;  
  
 **public void** setTransactionTemplate(TransactionTemplate transactionTemplate) {  
 **this**.**transactionTemplate** = transactionTemplate;  
 }  
  
 @Override  
 **public void** transfer(**final** String outer, **final** String inner, **final** Float money) {  
  
  
 *//扣钱* **accountDao**.out(outer,money);  
 **int** i=8/0;  
 *//进账* **accountDao**.in(inner,money);  
  
  
  
 }  
}

Bean:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xmlns:tx="http://www.springframework.org/schema/tx"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd"**>  
  
 <**bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>  
 *<!-- &lt;!&ndash;连接数据库的必备信息&ndash;&gt;-->* <**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="root"**></**property**>  
 </**bean**>  
  
 *<!--配置dao-->* <**bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl"**>  
 <**property name="dataSource" ref="dataSource"**></**property**>  
 </**bean**>  
  
  
 *<!--配置事务管理器-->* <**bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property name="dataSource" ref="dataSource"**></**property**>  
 </**bean**>  
  
  
  
 *<!--配置service-->* <**bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl2"**>  
 <**property name="accountDao" ref="accountDao"**></**property**>  
 </**bean**>  
  
 *<!--使用spring和aop标签来配置事务-->  
 <!--配置通知的事务管理器-->* <**tx:advice id="txAdvice" transaction-manager="txManager"**>  
 *<!--//事务的xianfq ,传播行为隔离级别可以不用配置-->* <**tx:attributes**>  
 <**tx:method name="transfer"**/>  
 </**tx:attributes**>  
 </**tx:advice**>  
  
 <**aop:config**>  
 <**aop:pointcut id="myPointCut" expression="execution(\* com.itheima.service..\*.\*(..))"**></**aop:pointcut**>  
 <**aop:advisor advice-ref="txAdvice" pointcut-ref="myPointCut"**></**aop:advisor**>  
 </**aop:config**>  
  
</**beans**>

**2.注解配置**

**ServiceImpl:**

@Transactional  
**public class** AccountServiceImpl3 **implements** IAccountService{  
  
 **private** IAccountDao **accountDao**;  
  
 **public void** setAccountDao(IAccountDao accountDao) {  
 **this**.**accountDao** = accountDao;  
 }  
  
  
  
 @Override  
 **public void** transfer(**final** String outer, **final** String inner, **final** Float money) {  
  
  
 *//扣钱* **accountDao**.out(outer,money);  
*// int i=8/0;  
 //进账* **accountDao**.in(inner,money);  
  
  
  
 }  
}

**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"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xmlns:tx="http://www.springframework.org/schema/tx"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd"**>  
  
 <**bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>  
 *<!-- &lt;!&ndash;连接数据库的必备信息&ndash;&gt;-->* <**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="root"**></**property**>  
 </**bean**>  
  
 *<!--配置dao-->* <**bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl"**>  
 <**property name="dataSource" ref="dataSource"**></**property**>  
 </**bean**>  
  
  
  
  
 *<!--配置service-->* <**bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl3"**>  
 <**property name="accountDao" ref="accountDao"**></**property**>  
 </**bean**>  
  
  
 *<!--注解的事务配置-->  
 <!--配置事务管理器-->* <**bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property name="dataSource" ref="dataSource"**></**property**>  
 </**bean**>  
  
 *<!--开启事务注解驱动-->* <**tx:annotation-driven transaction-manager="txManager"**></**tx:annotation-driven**>  
</**beans**>

**测试：**

**public class** AccountServiceTest2 {  
 @Test  
 **public void** test1(){  
 ApplicationContext context=**new** ClassPathXmlApplicationContext(**"bean3.xml"**);  
 *//获取service代理对象* IAccountService accountService = context.getBean(**"accountService"**, IAccountService.**class**);  
 System.***out***.println(accountService);  
 accountService.transfer(**"bbb"**,**"ccc"**,100f);  
 }  
  
}

**JDBCTemplate:**

*/\*\*  
 \* 账户的持久层实现类  
 \*/***public class** AccountDaoImpl **extends** JdbcDaoSupport **implements** IAccountDao {  
  
 @Override  
 **public void** out(String outer, Float money) {  
  
 **super**.getJdbcTemplate().update(**"update account set money=money-? where name=?"**, money,outer);  
 }  
  
 @Override  
 **public void** in(String inner, Float money) {  
 **super**.getJdbcTemplate().update(**"update account set money=money+? where name=?"**, money,inner);  
 }  
  
}