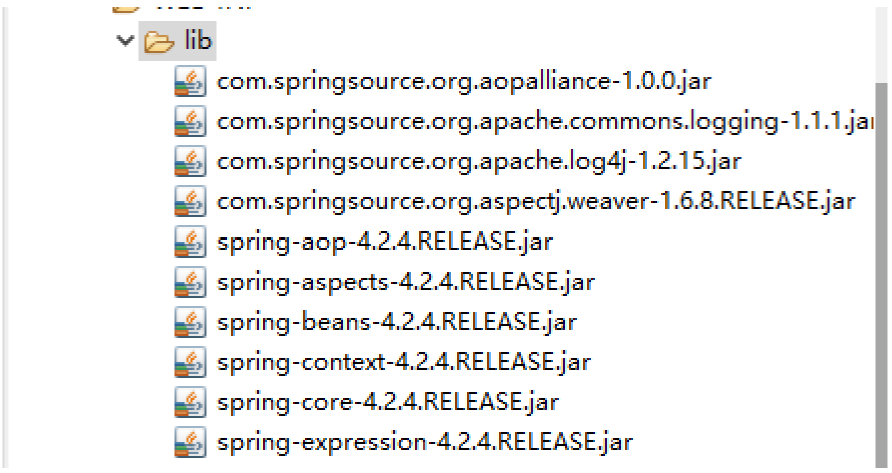
1.2 Spring的AOP的基于AspectJ注解开发

1.2.1 Spring的基于ApsectJ的注解的AOP开发

1.2.1.1 创建项目，引入jar包



com.springsource.org.aopalliance-1.0.0.jar
com.springsource.org.apache.commons.logging-1.1.1.jar
com.springsource.org.apache.log4j-1.2.15.jar
com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar
spring-aop-4.2.4.RELEASE.jar
spring-aspects-4.2.4.RELEASE.jar
spring-beans-4.2.4.RELEASE.jar
spring-context-4.2.4.RELEASE.jar
spring-core-4.2.4.RELEASE.jar
spring-expression-4.2.4.RELEASE.jar

1.2.1.2 引入配置文件

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

</beans>
```

1.2.1.3 编写目标类并配置

OrderDao.java

```java
/**
* @Title: OrderDao.java
* @Package com.admiral.spring.demo1
* @Description:
```

```java
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

public interface OrderDao {

    public void save();
    public void update();
    public void delete();
    public void find();
}
```

OrderDaoImpl.java

```java
/**
* @Title: OrderDaoImpl.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

public class OrderDaoImpl implements OrderDao {

    @Override
    public void save() {
        System.out.println("保存订单....");
    }

    @Override
    public void update() {
        System.out.println("修改订单....");
    }

    @Override
    public void delete() {
        System.out.println("删除订单....");
    }

    @Override
    public void find() {
        System.out.println("查询订单....");
    }

}
```

applicationContext.xml

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置目标对象,被增强的对象 -->
    <bean id="orderDao" class="com.admiral.spring.demo1.OrderDaoImpl"></bean>



</beans>
```

1.2.1.4 编写切面类并配置

MyAspect.java

```java
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

public class MyAspect {

    public void before() {
        System.out.println("前置增强 ==== ");
    }
}
```

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置目标对象,被增强的对象 -->
    <bean id="orderDao" class="com.admiral.spring.demo1.OrderDaoImpl"></bean>

    <!-- 配置切面类 -->
    <bean id="myAspect" class="com.admiral.spring.demo1.MyAspect"></bean>

</beans>
```

1.2.1.5 使用注解的AOP对目标类进行增强

- 在配置文件中打开注解的 AOP 开发

```
<?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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 开启注解的 AOP 开发 -->
    <aop:aspectj-autoproxy />

    <!-- 配置目标对象,被增强的对象 -->
    <bean id="orderDao" class="com.admiral.spring.demo1.OrderDaoImpl"></bean>

    <!-- 配置切面类 -->
    <bean id="myAspect" class="com.admiral.spring.demo1.MyAspect"></bean>
```

```
</beans>
```

- 在切面类上使用注解

```
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class MyAspect {

    @Before(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.save(..))")
    public void before() {
        System.out.println("前置增强 ==== ");
    }
}
```

1.2.1.6 编写测试类

```
/**
* @Title: SpringDemo1.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class SpringDemo1 {

    @Resource(name = "orderDao")
    private OrderDao orderDao;

    @Test
    public void demo1() {
        orderDao.save();
        orderDao.update();
        orderDao.delete();
        orderDao.find();
    }
}
```

1.2.2 Spring的注解的AOP的通知类型

1.2.2.1 @Before：前置通知

1.2.2.2 @AfterReturning：后置通知

```
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class MyAspect {

    @AfterReturning(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.delete(..))",returning = "result")
    public void afterReturning(Object result) {
        System.out.println("后置增强 ==== " + result);
    }
}
```

1.2.2.3 @Around：环绕通知

```
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class MyAspect {

    @Around(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.update(..))")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
        System.out.println("环绕前增强 ==== ");
        Object obj = joinPoint.proceed();
        System.out.println("环绕后增强 ==== ");
        return obj;
    }

}
```

1.2.2.4 @AfterThrowing：异常抛出通知

```
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class MyAspect {

    @AfterThrowing(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.find(..))",throwing = "ex")
    public void afterThrowing(Throwable ex) {
        System.out.println("异常抛出增强 ==== " + ex.getMessage());
    }

}
```

1.2.2.5 @After：最终通知

```
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class MyAspect {

    @After(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.find(..))")
    public void after() {
        System.out.println("最终增强 ==== ");
    }
}
```

1.2.3 Spring的注解的AOP的切入点的配置

1.2.3.1 Spring的AOP的注解切入点的配置

```
/**
* @Title: MyAspect.java
* @Package com.admiral.spring.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.spring.demo1;
```

```java
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class MyAspect {

    @Before(value = "MyAspect.pointcut1()")
    public void before() {
        System.out.println("前置增强 ==== ");
    }

    @AfterReturning(value = "MyAspect.pointcut3()",returning = "result")
    public void afterReturning(Object result) {
        System.out.println("后置增强 ==== " + result);
    }

    @Around(value = "MyAspect.pointcut2()")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
        System.out.println("环绕前增强 ==== ");
        Object obj = joinPoint.proceed();
        System.out.println("环绕后增强 ==== ");
        return obj;
    }

    @AfterThrowing(value = "MyAspect.pointcut4()",throwing = "ex")
    public void afterThrowing(Throwable ex) {
        System.out.println("异常抛出增强 ==== " + ex.getMessage());
    }

    @After(value = "MyAspect.pointcut4()")
    public void after() {
        System.out.println("最终增强 ==== ");
    }

    @Pointcut(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.save(..))")
    public void pointcut1() {}
    @Pointcut(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.update(..))")
    public void pointcut2() {}
    @Pointcut(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.delete(..))")
    public void pointcut3() {}
    @Pointcut(value = "execution(*
com.admiral.spring.demo1.OrderDaoImpl.find(..))")
    public void pointcut4() {}
}
```

1.3 Spring的JDBC的模板的使用

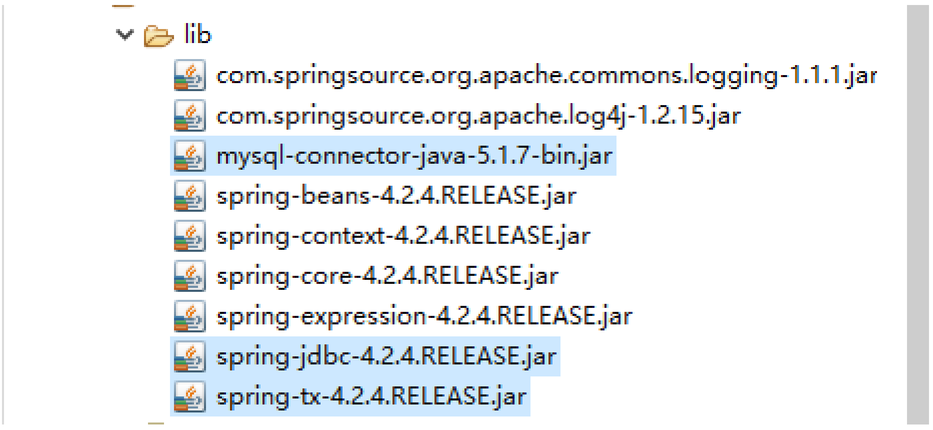
1.3.1 Spring的JDBC的模板

Spring是EE开发的一站式的框架，有EE开发的每层的解决方案。Spring对持久层也提供了解决方案：ORM模块和JDBC的模板。
Spring提供了很多的模板用于简化开发：

JIDBC	org.springframework.jdbc.core.JdbcTemplate
Hibernate3. 0	org.springframeworkorm. hibernates. HibernateTemplate
IBatis(MyBatis)	org.springframework.orm.ibatis.SqlMapClientTemplate
JPA	org.springframework.orm.jpa.JpaTemplate

1.3.1.1 JDBC模板使用的入门

- 创建项目，引入jar包
  - 引入基本开发包：
  - 数据库驱动
  - Spring的JDBC模板的jar包



```
lib
    com.springsource.org.apache.commons.logging-1.1.1.jar
    com.springsource.org.apache.log4j-1.2.15.jar
    mysql-connector-java-5.1.7-bin.jar
    spring-beans-4.2.4.RELEASE.jar
    spring-context-4.2.4.RELEASE.jar
    spring-core-4.2.4.RELEASE.jar
    spring-expression-4.2.4.RELEASE.jar
    spring-jdbc-4.2.4.RELEASE.jar
    spring-tx-4.2.4.RELEASE.jar
```

1.3.1.2 创建数据库和表

```sql
create database spring4_day03;
use spring4_day03;
create table account(
    id int primary key auto_increment,
    name varchar(20),
    money double
);
```

1.3.1.3 使用JDBC的模板：保存数据

```java
/**
* @Title: JdbcDemo1.java
* @Package com.admiral.jdbc.demo1
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.jdbc.demo1;

import org.junit.Test;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

public class JdbcDemo1 {

    @Test
    public void demo1() {
        //创建数据库连接池
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql:///spring4_day03");
        dataSource.setUsername("root");
        dataSource.setPassword("111111");
        //创建 Jdbc 模板
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.update("insert into account values(null,?,?)", "小红
红",10000d);

    }
}
```

1.3.2 将连接池和模板交给Spring管理

1.3.2.1 引入Spring的配置文件

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">
    <!-- 配置 Spring 内置的连接池 -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql:///spring4_day03" />
        <property name="username" value="root" />
        <property name="password" value="111111" />
    </bean>

    <!-- 配置 Spring 的 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

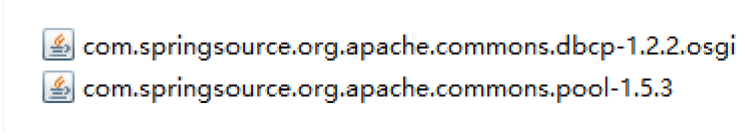
1.3.2.2 使用Jdbc的模板

```java
/**
* @Title: JdbcDemo2.java
* @Package com.admiral.jdbc.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.jdbc.demo2;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JdbcDemo2 {

    @Resource(name = "jdbcTemplate")
    private JdbcTemplate JdbcTemplate;

    @Test
    public void demo1() {
        JdbcTemplate.update("insert into account values(null,?,?)", "小黑
黑",20000d);
    }
}
```

1.3.3 使用开源的数据库连接池：

1.3.3.1 DBCP的使用

- 引入 jar 包



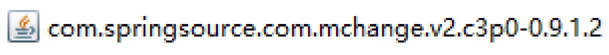
com.springsource.org.apache.commons.dbcp-1.2.2.osgi
com.springsource.org.apache.commons.pool-1.5.3

- 配置 DBCP 连接池

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 DBCP 连接池 -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql:///spring4_day03" />
        <property name="username" value="root" />
        <property name="password" value="111111" />
    </bean>

    <!-- 配置 Spring 的 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

1.3.3.2 C3P0的使用

- 引入 C3P0 连接池的 jar 包



com.springsource.com.mchange.v2.c3p0-0.9.1.2

- 配置 C3P0 连接池

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql:///spring4_day03" />
        <property name="user" value="root" />
        <property name="password" value="111111" />
    </bean>

    <!-- 配置 Spring 的 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

1.3.4 抽取配置到属性文件

1.3.4.1 定义一个属性文件

```
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///spring4_day03
jdbc.username=root
jdbc.password=111111
```

1.3.4.2 在Spring的配置文件中引入属性文件

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">


    <!-- 方式一:引入外部属性文件 -->
    <bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="classpath:jdbc.properties" />
    </bean>
    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

</beans>
```

1.3.4.3 引入属性文件的值

```
<?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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 方式一:引入外部属性文件 -->
<!--    <bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="classpath:jdbc.properties" />
    </bean> -->
    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 配置 Spring 的 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
```

```
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

1.3.4.4 测试

```
/**
* @Title: JdbcDemo2.java
* @Package com.admiral.jdbc.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.jdbc.demo2;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JdbcDemo2 {

    @Resource(name = "jdbcTemplate")
    private JdbcTemplate JdbcTemplate;

    @Test
    public void demo1() {
        JdbcTemplate.update("insert into account values(null,?,?)", "小绿
绿",40000d);
    }
}
```

1.3.5 使用JDBC的模板完成CRUD的操作

1.3.5.1 保存操作

```
/**
* @Title: JdbcDemo2.java
* @Package com.admiral.jdbc.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
```

```
package com.admiral.jdbc.demo2;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JdbcDemo2 {

    @Resource(name = "jdbcTemplate")
    private JdbcTemplate JdbcTemplate;

    @Test
    // 保存操作
    public void demo1() {
        JdbcTemplate.update("insert into account values(null,?,?)", "小绿
绿",40000d);
    }

}
```

1.3.5.2 修改操作

```
/**
* @Title: JdbcDemo2.java
* @Package com.admiral.jdbc.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.jdbc.demo2;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JdbcDemo2 {

    @Resource(name = "jdbcTemplate")
    private JdbcTemplate JdbcTemplate;

    @Test
    // 保存操作
    public void demo1() {
        JdbcTemplate.update("insert into account values(null,?,?)", "小绿
绿",40000d);
    }
    @Test
    // 修改操作
    public void demo2() {
        JdbcTemplate.update("update account set name=? where id=?", "小紫紫",3);
    }
}
```

1.3.5.3 删除操作

```
/**
* @Title: JdbcDemo2.java
* @Package com.admiral.jdbc.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.jdbc.demo2;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JdbcDemo2 {

    @Resource(name = "jdbcTemplate")
    private JdbcTemplate JdbcTemplate;

    @Test
    // 保存操作
    public void demo1() {
        JdbcTemplate.update("insert into account values(null,?,?)", "小绿
绿",40000d);
    }
    @Test
    // 修改操作
    public void demo2() {
        JdbcTemplate.update("update account set name=? where id=?", "小紫紫",3);
    }
    @Test
    // 删除操作
    public void demo3() {
        JdbcTemplate.update("delete from account where id=?", 4);
    }
}
```

1.3.5.4 查询操作

```
/**
* @Title: JdbcDemo2.java
* @Package com.admiral.jdbc.demo2
* @Description:
* @author 白世鑫
* @date 2020-10-12
* @version V1.0
*/
package com.admiral.jdbc.demo2;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.admiral.jdbc.domain.Account;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JdbcDemo2 {

    @Resource(name = "jdbcTemplate")
    private JdbcTemplate JdbcTemplate;

    @Test
    // 查询返回对象
    public void demo6() {
        Account account = JdbcTemplate.queryForObject("select * from account
where id=?", new MyRowMapper(), 7);
        System.out.println(account);
    }

    @Test
    // 查询返回集合
    public void demo7() {
        List<Account> accounts = JdbcTemplate.query("select * from account", new
MyRowMapper());
        for (Account account : accounts) {
            System.out.println(account);
        }
    }
    class MyRowMapper implements RowMapper<Account>{

        @Override
        public Account mapRow(ResultSet rs, int rowNum) throws SQLException {
            Account account = new Account();
            account.setId(rs.getInt("id"));
            account.setName(rs.getString("name"));
            account.setMoney(rs.getDouble("money"));
            return account;
        }

    }
}
```

1.4 Spring的事务管理

1.4.1 事务的回顾

1.4.1.1 什么是事务

- 事务：逻辑上的一组操作，组成这组操作的各个单元，要么全都成功，要么全都失败。

1.4.1.2 事务的特性

- 原子性：事务不可分割
- 一致性：事务执行前后数据完整性保持一致
- 隔离性：一个事务的执行不应该受到其他事务的干扰
- 持久性：一旦事务结束，数据就持久化到数据库

1.4.1.3 如果不考虑隔离性引发安全性问题

- 读问题
    - 脏读 ：一个事务读到另一个事务未提交的数据
    - 不可重复读 ：一个事务读到另一个事务已经提交的update的数据，导致一个事务中多次查询结果不一致
    - 虚读、幻读 ：一个事务读到另一个事务已经提交的insert的数据，导致一个事务中多次查询结果不一致。
- 写问题
    - 丢失更新

1.4.1.4 解决读问题

- 设置事务的隔离级别
    - Read uncommitted ：未提交读，任何读问题解决不了。
    - **Read committed ：已提交读，解决脏读，但是不可重复读和虚读有可能发生。**
    - **Repeatable read ：重复读，解决脏读和不可重复读，但是虚读有可能发生。**
    - Serializable ：解决所有读问题。

1.4.2 Spring的事务管理的API

1.4.2.1 PlatformTransactionManager：平台事务管理器

- 平台事务管理器：接口，是Spring用于管理事务的真正的对象。
    - DataSourceTransactionManager ：底层使用JDBC管理事务
    - HibernateTransactionManager ：底层使用Hibernate管理事务

1.4.2.2 TransactionDefinition ：事务定义信息

- 事务定义：用于定义事务的相关的信息，隔离级别、超时信息、传播行为、是否只读

1.4.2.3 TransactionStatus：事务的状态

- 事务状态：用于记录在事务管理过程中，事务的状态的对象。

1.4.2.4 事务管理的API的关系：

Spring进行事务管理的时候，首先平台事务管理器根据事务定义信息进行事务的管理，在事务管理过程中，产生各种状态，将这些状态的信息记录到事务状态的对象中。

1.4.3 Spring的事务的传播行为

1.4.3.1 Spring的传播行为

> 保证同一个事务中
>
> PROPAGATION_REQUIRED 支持当前事务,如果不存在就新建一个（默认）
>
> PROPAGATION_SUPPORTS 支持当前事务,如果不存在，就不使用事务
>
> PROPAGATION_MANDATORY 支持当前事务,如果不存在，抛出异常
>
> 保证没有在同一个事务中
>
> PROPAGATION_REQUIRES_NEW 如果有事务存在，挂起当前事务，创建一个新的事务
>
> ROPAGATION_NOT_SUPPORTED 以非事务方式运行，如果有事务存在，挂起当前事务
>
> PROPAGATION_NEVER 以非事务方式运行，如果有事务存在，抛出异常
>
> 嵌套事务
>
> PROPAGATION_NESTED 如果当前事务存在，则嵌套事务执行

- Spring中提供了七种事务的传播行为：
    - 保证多个操作在同一个事务中
        - **PROPAGATION_REQUIRED ：默认值，如果A中有事务，使用A中的事务，如果A没有，创建一个新的事务，将操作包含进来**
        - PROPAGATION_SUPPORTS ：支持事务，如果A中有事务，使用A中的事务。如果A没有事务，不使用事务。
        - PROPAGATION_MANDATORY ：如果A中有事务，使用A中的事务。如果A没有事务，抛出异常。

- 保证多个操作不在同一个事务中
  - **PROPAGATION_REQUIRES_NEW　：如果A中有事务，将A的事务挂起（暂停），创建新事务，只包含自身操作。如果A中没有事务，创建一个新事务，包含自身操作。**
  - PROPAGATION_NOT_SUPPORTED　：如果A中有事务，将A的事务挂起。不使用事务管理。
  - PROPAGATION_NEVER　　　　：如果A中有事务，报异常。
- 嵌套式事务
  - **PROPAGATION_NESTED　　　：嵌套事务，如果A中有事务，按照A的事务执行，执行完成后，设置一个保存点，执行B中的操作，如果没有异常，执行通过，如果有异常，可以选择回滚到最初始位置，也可以回滚到保存点。**

1.4.4 Spring的事务管理

1.4.4.1 搭建Spring的事务管理的环境

- 创建Service的接口和实现类

```java
/**
* @Title: AccountService.java
* @Package com.admiral.jdbc.demo3
* @Description: 转账的业务层接口
* @author 白世鑫
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;

public interface AccountService {

    /**
     *
     * Title: transfer
     * Description:
     * @param from  :转出账号
     * @param to    :转入账号
     * @param money :转账金额
     */
    public void transfer(String from, String to, Double money);
}
```

```java
/**
* @Title: AccountServiceImpl.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;
```

```java
public class AccountServiceImpl implements AccountService {

    //注入 Dao
    private AccountDao accountDao;

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

    @Override
    public void transfer(String from, String to, Double money) {

    }

}
```

- 创建DAO的接口和实现类

```java
/**
* @Title: AccountDao.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;

public interface AccountDao {

    /**
     *
     * Title: outMoney
     * Description:
     * @param from  :转出账号
     * @param money :转出金额
     */
    public void outMoney(String from,Double money);

    /**
     *
     * Title: inMoney
     * Description:
     * @param to    :转入账号
     * @param money :转账金额
     */
    public void inMoney(String to,Double money);
}
```

```java
/**
* @Title: AccountDaoImpl.java
* @Package com.admiral.jdbc.demo3
* @Description:
```

```
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;

public class AccountDaoImpl implements AccountDao {

    @Override
    public void outMoney(String from, Double money) {

    }

    @Override
    public void inMoney(String to, Double money) {

    }

}
```

- 配置Service和DAO：交给Spring管理

```
<?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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.jdbc.demo3.AccountDaoImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.jdbc.demo3.AccountDaoImpl"></bean>
</beans>
```

- 在DAO中编写扣钱和加钱方法：
    - 配置连接池和 JDBC 模板

```
<?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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.jdbc.demo3.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.jdbc.demo3.AccountDaoImpl">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>
</beans>
```

```
/**
* @Title: AccountDaoImpl.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;

import org.springframework.jdbc.core.support.JdbcDaoSupport;

public class AccountDaoImpl extends JdbcDaoSupport implements AccountDao {


    @Override
    public void outMoney(String from, Double money) {
        this.getJdbcTemplate().update("update account set money = money - ?
where name = ?", money,from);
    }

    @Override
    public void inMoney(String to, Double money) {
        this.getJdbcTemplate().update("update account set money = money + ?
where name = ?", money,to);
    }

}
```

- 测试

```
/**
* @Title: SpringDemo3.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:tx.xml")
public class SpringDemo3 {

    @Resource(name = "accountService")
    private AccountService accountService;

    @Test
    public void demo1() {
        accountService.transfer("小黑黑", "小黄黄", 500d);
    }
}
```

1.4.5 Spring的事务管理：一类：编程式事务（需要手动编写代码）--了解

1.4.5.1 第一步：配置平台事务管理器

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>
```

1.4.5.2 第二步：Spring提供了事务管理的模板类

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置事务管理的模板类 -->
    <bean id="transactionTemplate"
class="org.springframework.transaction.support.TransactionTemplate">
```

```
        <property name="transactionManager" ref="transactionManager" />
    </bean>

</beans>
```

1.4.5.3 第三步：在业务层注入事务管理的模板

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.jdbc.demo3.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
        <!-- 注入事务管理模板类 -->
        <property name="transactionTemplate" ref="transactionTemplate" />
    </bean>

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置事务管理的模板类 -->
    <bean id="transactionTemplate"
class="org.springframework.transaction.support.TransactionTemplate">
        <property name="transactionManager" ref="transactionManager" />
    </bean>

</beans>
```

1.4.5.4 编写事务管理的代码

```java
/**
* @Title: AccountServiceImpl.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
```

```java
*/
package com.admiral.jdbc.demo3;

import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.TransactionCallbackWithoutResult;
import org.springframework.transaction.support.TransactionTemplate;

public class AccountServiceImpl implements AccountService {

    // 注入 Dao
    private AccountDao accountDao;

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

    // 注入事务管理模板类
    private TransactionTemplate transactionTemplate;

    public void setTransactionTemplate(TransactionTemplate transactionTemplate)
{
        this.transactionTemplate = transactionTemplate;
    }

    @Override
    public void transfer(String from, String to, Double money) {

        transactionTemplate.execute(new TransactionCallbackWithoutResult() {

            @Override
            protected void doInTransactionWithoutResult(TransactionStatus
transactionStatus) {
                accountDao.outMoney(from, money);
                int i = 1 / 0;
                accountDao.inMoney(to, money);
            }
        });
    }
}
```

1.4.5.5 测试：

```java
/**
* @Title: SpringDemo3.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.jdbc.demo3;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:tx.xml")
public class SpringDemo3 {

    @Resource(name = "accountService")
    private AccountService accountService;

    @Test
    public void demo1() {
        accountService.transfer("小黑黑", "小黄黄", 1000d);
    }
}
```

1.4.6 Spring的事务管理：二类：声明式事务管理（通过配置实现）---AOP

1.4.6.1 XML方式的声明式事务管理

- 第一步：引入aop的开发包
- 第二步：恢复转账环境
- 第三步：配置事务管理器

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.tx.demo2.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.tx.demo2.AccountDaoImpl">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>
    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

- 第四步：配置增强

```
<?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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.tx.demo2.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.tx.demo2.AccountDaoImpl">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
```

```
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置事务的增强 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 事务管理的规则 -->
            <!-- <tx:method name="save*" propagation="REQUIRED"
isolation="DEFAULT"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="find*" read-only="true"/> -->
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>

</beans>
```

- 第五步：AOP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"
    xmlns:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.tx.demo2.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.tx.demo2.AccountDaoImpl">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>
    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置事务的增强 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 事务管理的规则 -->
            <!-- <tx:method name="save*" propagation="REQUIRED"
isolation="DEFAULT"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="find*" read-only="true"/> -->
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>

    <!-- AOP 的配置 -->
    <aop:config>
        <aop:pointcut expression="execution(*
com.admiral.tx.demo2.AccountServiceImpl.*(..))" id="pointcut1"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="pointcut1"/>
    </aop:config>
</beans>
```

- 测试

```
/**
* @Title: SpringDemo3.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.tx.demo2;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:tx2.xml")
public class SpringDemo3 {

    @Resource(name = "accountService")
    private AccountService accountService;

    @Test
    public void demo1() {
        accountService.transfer("小黑黑", "小黄黄", 1000d);
    }
}
```

1.4.6.2 注解方式的声明式事务管理

- 第一步：引入aop的开发包
- 第二步：恢复转账环境
- 第三步：配置事务管理器

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.tx.demo3.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.tx.demo3.AccountDaoImpl">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>
```

- 第四步:开启注解事务

```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:context="http://www.springframework.org/schema/context"
    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/context
    http://www.springframework.org/schema/context/spring-context.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">

    <!-- 配置 AccountService -->
    <bean id="accountService" class="com.admiral.tx.demo3.AccountServiceImpl">
        <property name="accountDao" ref="accountDao" />
    </bean>
    <!-- 配置 AccountDao -->
    <bean id="accountDao" class="com.admiral.tx.demo3.AccountDaoImpl">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 方式二:引入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置 C3P0 连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${jdbc.driverClass}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 配置平台事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 开启注解事务 -->
    <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

- 第五步:在业务层添加注解

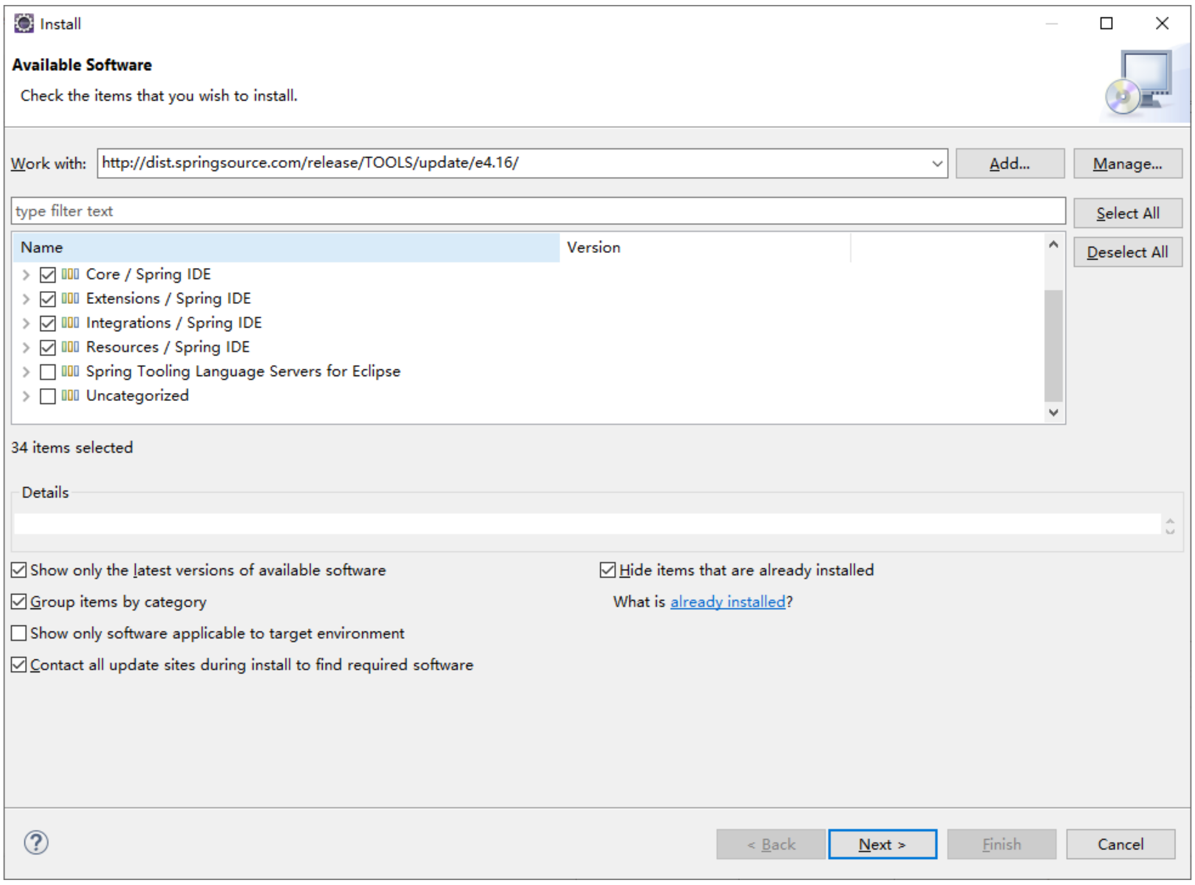
```java
/**
* @Title: AccountServiceImpl.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
```

```java
package com.admiral.tx.demo3;

import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@Transactional(isolation = Isolation.DEFAULT,propagation = Propagation.REQUIRED)
public class AccountServiceImpl implements AccountService {

    // 注入 Dao
    private AccountDao accountDao;

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

    @Override
    public void transfer(String from, String to, Double money) {

        accountDao.outMoney(from, money);
        int i = 1 / 0;
        accountDao.inMoney(to, money);

    }

}
```

- 测试

```java
/**
* @Title: SpringDemo3.java
* @Package com.admiral.jdbc.demo3
* @Description:
* @author Admiral
* @date 2020-10-13
* @version V1.0
*/
package com.admiral.tx.demo3;

import javax.annotation.Resource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:tx3.xml")
public class SpringDemo3 {

    @Resource(name = "accountService")
    private AccountService accountService;

    @Test
    public void demo1() {
        accountService.transfer("小黑黑", "小黄黄", 1000d);
    }
}
```

http://dist.springsource.com/release/TOOLS/update/e4.16/


Install

Available Software

Check the items that you wish to install.

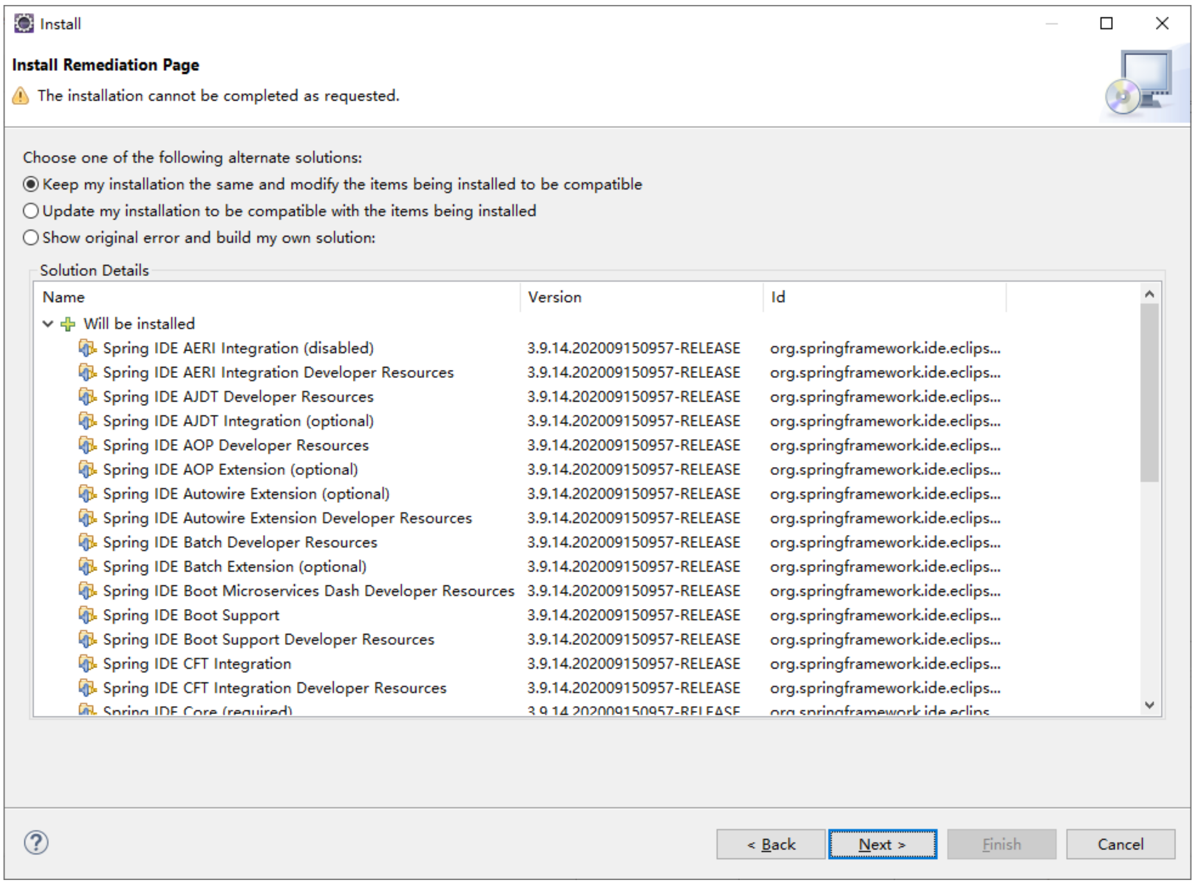
Work with: http://dist.springsource.com/release/TOOLS/update/e4.16/

Name	Version
☑ Core / Spring IDE	
☑ Extensions / Spring IDE	
☑ Integrations / Spring IDE	
☑ Resources / Spring IDE	
☐ Spring Tooling Language Servers for Eclipse	
☐ Uncategorized	

34 items selected

Details

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?



Install

Install Remediation Page

The installation cannot be completed as requested.

Choose one of the following alternate solutions:

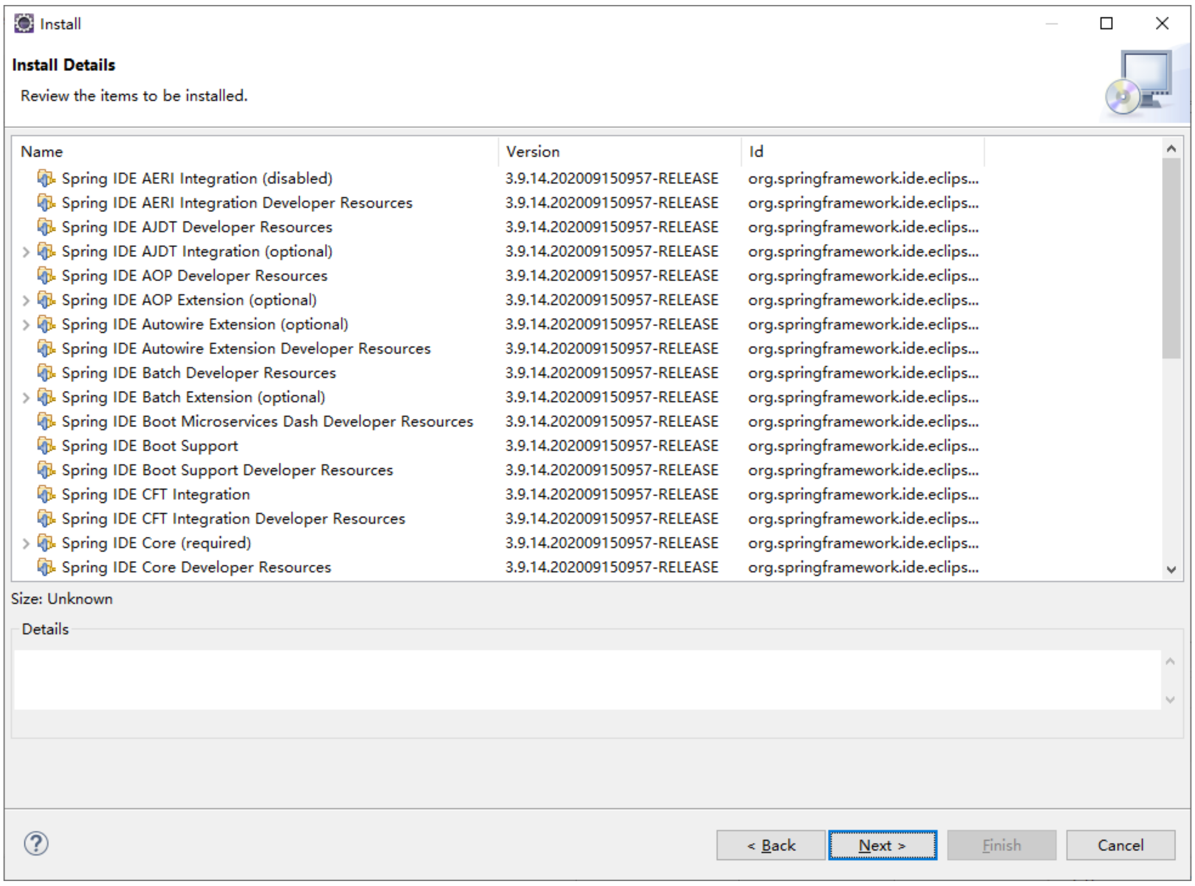
[✓ Keep my installation the same and modify the items being installed to be compatible]

Update my installation to be compatible with the items being installed

Show original error and build my own solution:

Solution Details

Name	Version	Id
Will be installed		
Spring IDE AERI Integration (disabled)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AERI Integration Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AJDT Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AJDT Integration (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AOP Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AOP Extension (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Autowire Extension (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Autowire Extension Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Batch Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Batch Extension (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Boot Microservices Dash Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Boot Support	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Boot Support Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE CFT Integration	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE CFT Integration Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Core (required)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...



Install

Install Details

Review the items to be installed.

Name	Version	Id
Spring IDE AERI Integration (disabled)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AERI Integration Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AJDT Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AJDT Integration (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AOP Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE AOP Extension (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Autowire Extension (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Autowire Extension Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Batch Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Batch Extension (optional)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Boot Microservices Dash Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Boot Support	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Boot Support Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE CFT Integration	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE CFT Integration Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Core (required)	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...
Spring IDE Core Developer Resources	3.9.14.202009150957-RELEASE	org.springframework.ide.eclips...

Size: Unknown

Details