* 学习目标

* 能够掌握Spring 整合单元测试框架

* 添加依赖--

* @RunTest（SpringJunit4Runner.class）
@ContextConfiguration("classpath:applicationContext.xml")

* @SpringJunitConfig(SpringConfiguration.class);

* Junit单元测试用可以用到Spring IOC

* 能够掌握Spring的AOP的概述

* AOP：Aspect(切入点和通知)+Oriented+Programming--横切思想去考虑问题的

* 在不修改源码的情况，增强功能

* 符合我们开闭原则，对修改进行关闭，对扩展进行

* 连接点---切入点---通知--织入（静态织入，动态织入）--代理模式--JDK代理模式和Cglib子类代理

* 面向编程的时候

* 怎么样找到我们的切入点：execution(修饰符 返回值 包.类.方法（参数类型）)---> 简单形式

* 编写通知：before,after,arround afterreturning afterthrowing

* 能够掌握SpringAOP的XML开发方式

* 编写一个切面

* 通知

* 把切面放到IOC中

* xml里面：编写切入点和通知对应起来

* 能够掌握SpringAOP的注解开发方式

* Spring2x

* 切面：通知+切入点

* @Aspect，@Pointcut,@Before,....

* xml--<aop:aspectj-autoproxy proxy-target-class="true"/>

* Spring3

* 切面：通知+切入点

* @Aspect，@Pointcut,@Before,....

* @EnableAspectJAutoProxy

* 能够了解Spring 集成JDBC

* 添加依赖-复制代码--看一下效果

---

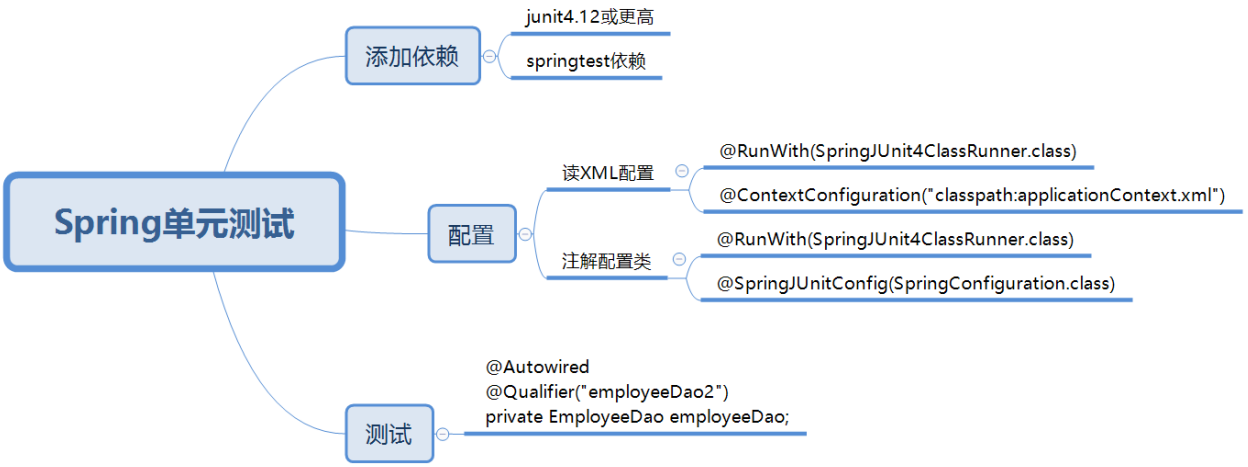
* 回顾

* Spring：2003

* IOC，AOP，SpEL，JDBC，MyBatis，SpringMVC，...

* IOC:控制反转

* 能够掌握Spring 整合单元测试框架



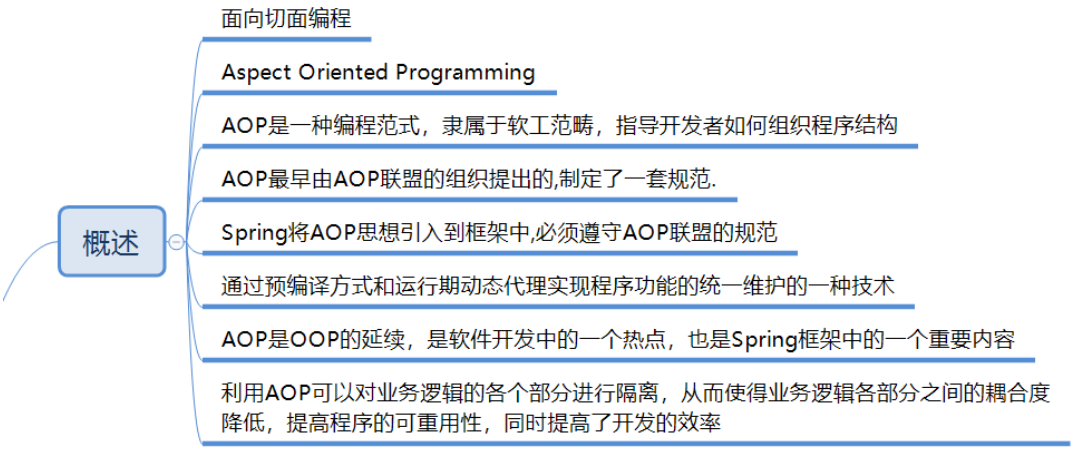

```
* 添加依赖
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
```

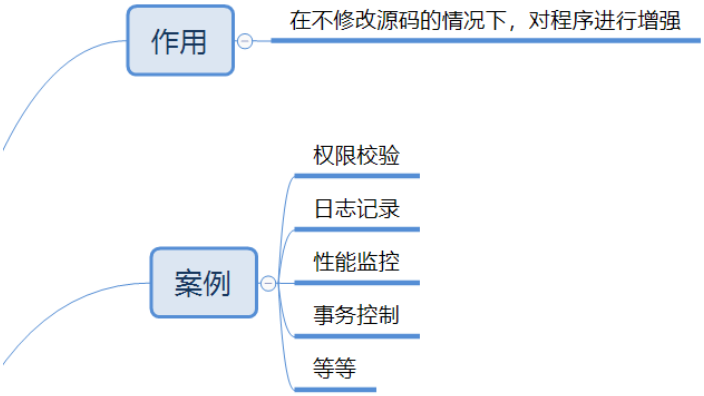
```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.10</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.2.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>5.2.2.RELEASE</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
* 单元测试代码
@RunWith(SpringJUnit4ClassRunner.class)
/*@ContextConfiguration("classpath:applicationContext.xml")*/
@SpringJUnitConfig(SpringConfiguration.class)
public class AppTest3
{
    @Autowired
    @Qualifier("employeeDao2")
    private EmployeeDao employeeDao;
    @Test
    public void test1(){
        employeeDao.addEmployee(new Employee());
    }
}
 * 温馨提醒：Spring单元测试，就可以直接使用IOC容器的对象
```

* 能够掌握Spring的AOP的概述

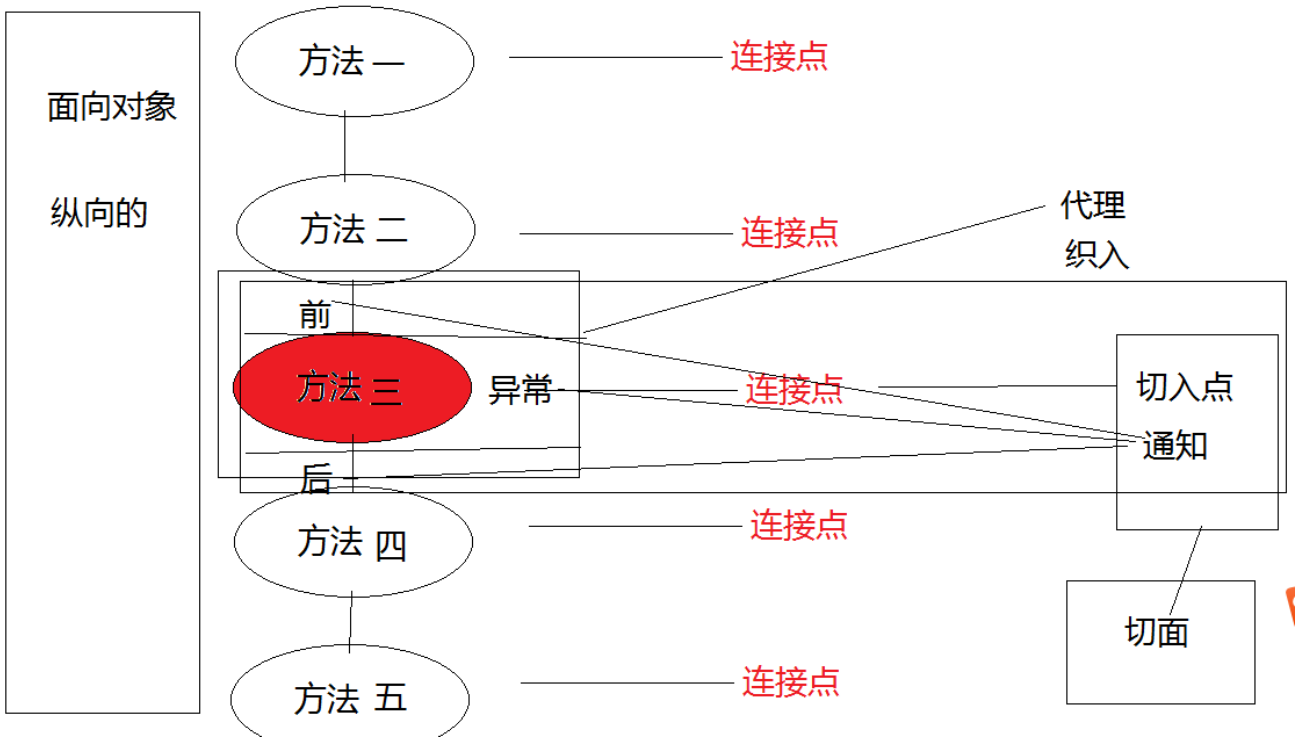
* 复习代理： 09-JDBC高级新1

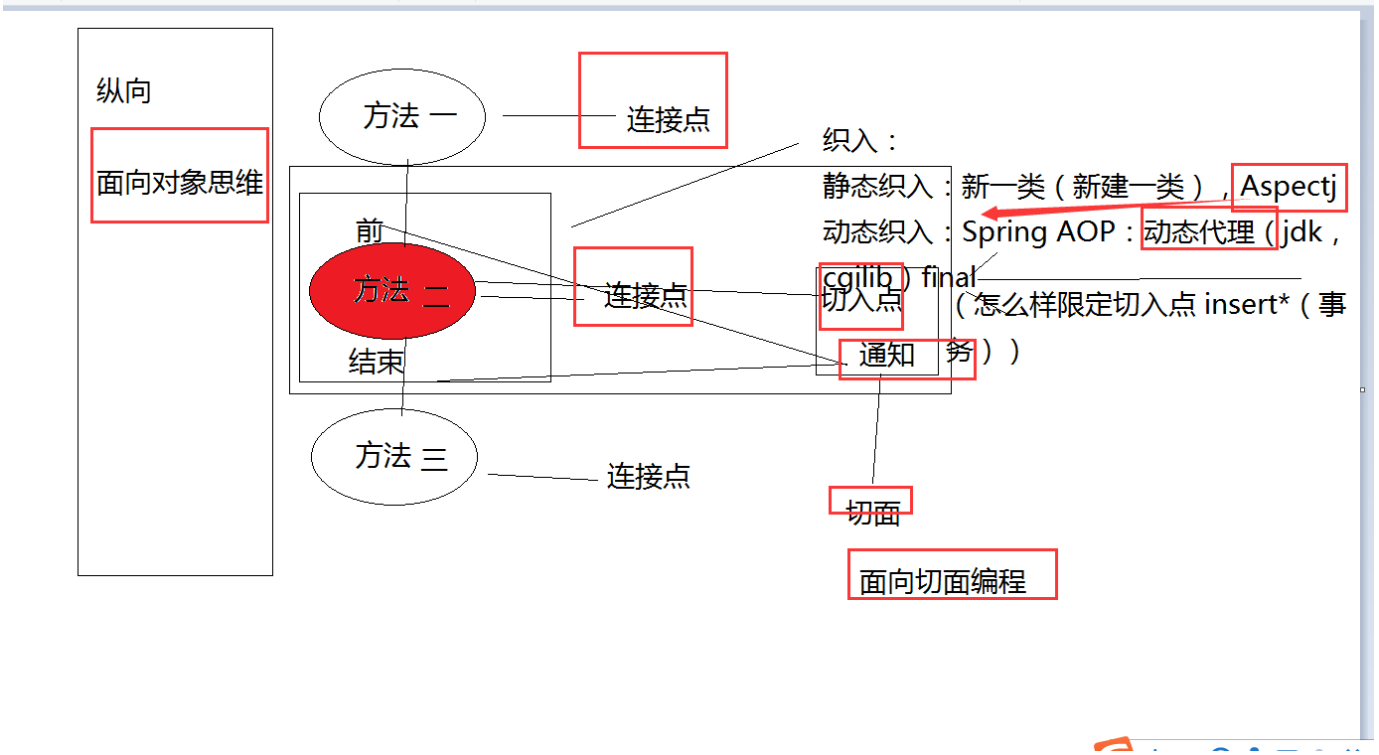
* AOP的概述


概述
面向切面编程
Aspect Oriented Programming
AOP是一种编程范式，隶属于软工范畴，指导开发者如何组织程序结构
AOP最早由AOP联盟的组织提出的,制定了一套规范.
Spring将AOP思想引入到框架中,必须遵守AOP联盟的规范
通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术
AOP是OOP的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容
利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率

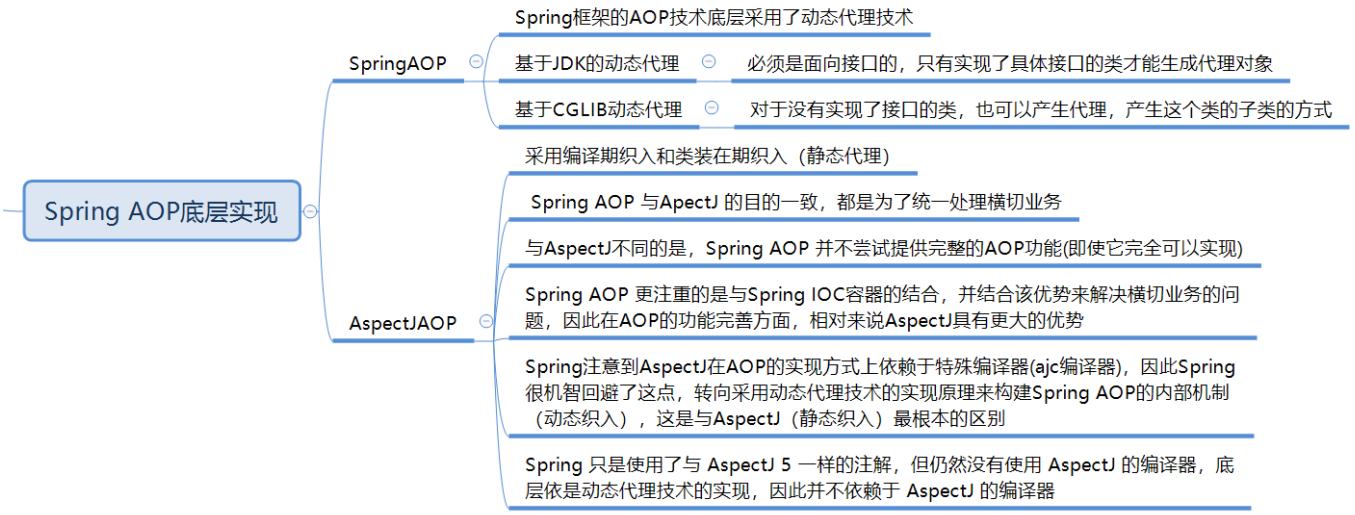

作用
在不修改源码的情况下，对程序进行增强
案例
权限校验
日志记录
性能监控
事务控制
等等

* 画图理解AOP


面向对象
纵向的
方法 一
连接点
方法 二
连接点
代理
织入
前
方法 三
异常
连接点
切入点
通知
后
方法 四
连接点
切面
方法 五
连接点

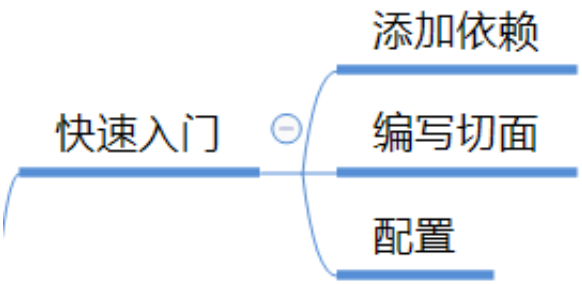


* AOP底层实现



* AOP相关的术语

* 能够掌握Spring的AOP的XML开发方式




```
* 案例一（xml）：快速入门
 * 添加依赖
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>5.2.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.9.5</version>
        </dependency>
        <dependency>
            <groupId>aopalliance</groupId>
            <artifactId>aopalliance</artifactId>
            <version>1.0</version>
        </dependency>
* 代码
public interface UserDao {
    public void addUser(User user);
}
@Repository("userDao")
public class UserDaoImpl implements UserDao {
    @Override
    public void addUser(User user) {
        System.out.println("addUser");
    }
```

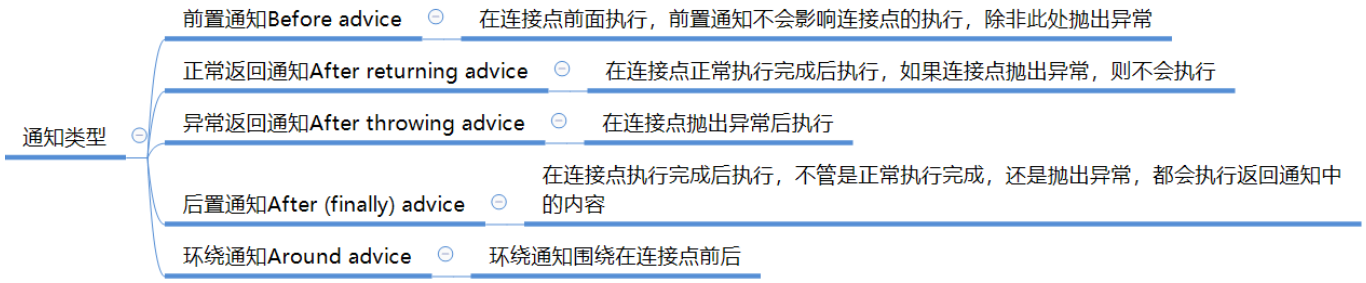
```
}
* 切面
public class TransactionAspect {
    public void startTransaction(){
        System.out.println("开始事务...");
    }

    public void commitTransaction(){
        System.out.println("提交事务...");
    }
}
* 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"
       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
    <context:component-scan base-package="com.lg.*"/>
    <bean id="userDao" class="com.lg.dao.impl.UserDaoImpl"/>
    <bean id="tsa" class="com.lg.aspect.TransactionAspect"/>
    <aop:config>
     <aop:aspect ref="tsa">
       <aop:before method="startTransaction" pointcut="execution(public void co
           (com.lg.bean.User))"/>
       <aop:after method="commitTransaction" pointcut="execution(public void co
           (com.lg.bean.User))"/>
     </aop:aspect>
    </aop:config>
* 单元测试
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
/*@SpringJUnitConfig(SpringConfiguration.class)*/
public class AppTest4
{
    @Autowired
```

```
68      private UserDao userDao;
69      @Test
70      public void test1(){
71          userDao.addUser(new User());
72      }
73  }
74
```

* execution表达式简介


```
1  * 案例二：简化表达式编写
2  <aop:aspect ref="tsa">
3    <aop:before method="startTransaction" pointcut="execution(* addUser(..))"></a
4    <aop:after method="commitTransaction" pointcut="execution(* addUser(..))"></a
5  </aop:aspect>
```

* 通知类型



```
* 案例：测试通知类型
* 代码
public interface ISpeakService {
     void speakEnligh();
     String speak(String languge);
     void err(boolean isThrow);
}
@Service("speackService")
public class SpeakServiceImpl implements ISpeakService {
    @Override
    public void speakEnligh() {
        System.out.println("speak English");
    }
    @Override
    public String speak(String languge) {
        System.out.println("speak "+languge);
        return languge;
    }
    @Override
    public void err(boolean isThrow) {
        System.out.println("error....");
        throw new RuntimeException("speak error");
    }
}
* 切面
@Component("sa")
public class SpeakAspect {
    public void speakBefore(){
        System.out.println("---speakBefore---");
    }
    public Object speakAround(ProceedingJoinPoint pjp) throws Throwable {
        Object obj=null;
        System.out.println("---around前---");
        try{
            // 执行目标方法
            obj = pjp.proceed();
        }catch (Throwable e){
            System.out.println("---around异常---");
            // 监听参数为true则抛出异常，为false则捕获并不抛出异常
            if(pjp.getArgs().length>0 && (Boolean) pjp.getArgs()[0]){
```

```
                throw e;
            }else{
                obj=null;
            }
        }
        System.out.println("---around后---");
        return obj;
    }
    public void speakAfter(){
        System.out.println("---speakAfter---");
    }
    public void speakAfterThrowing(){
        System.out.println("speakAfterThrowing");
    }
    public void speakAfterReturning(){
        System.out.println("speakAfeterReturning");
    }
}

* 配置
 <aop:config>
   <aop:aspect ref="sa">
     <!--定义连接点-->
     <aop:pointcut id="speak" expression="execution(* com.lg.service.impl.Speak
     <aop:before method="speakBefore" pointcut-ref="speak"/>
     <aop:around method="speakAround" pointcut-ref="speak"/>
     <aop:after method="speakAfter" pointcut-ref="speak"/>
     <aop:after-returning method="speakAfterReturning" pointcut-ref="speak"/>
     <aop:after-throwing method="speakAfterThrowing" pointcut-ref="speak"/>
   </aop:aspect>
 </aop:config>
 * 单元测试
  @Autowired
private ISpeakService speakService;
  @Test
 public void test2(){
  speakService.speakEnligh();
 }
 * 结果：speakBefore--around前--speak English--around后--speakAfter-- speakAfete
 @Test
```

```
  public void test3(){
    String language = speakService.speak("中文");
    System.out.println(language);
   }
 * 结果：speakBefore--around前--speak 中文--around后--speakAfter-- speakAfeterRe
  @Test
 public void test4(){
   speakService.err(false);
   }
 * 结果：speakBefore--around前--error--around异常--around后--speakAfter-- speakA
  @Test
  public void test5(){
    speakService.err(true);
  }
  * 结果：speakBefore--around前--error--around异常--speakAfter-- speakAfterThrow
  * 报异常
```

* 能够掌握SpringAOP的注解开发方式

```
* 案例：测试
* 配置方式一
<!--proxy-target-class为false或者省略这个属性基于jdk接口代理（假如没有接口类，会自动
<!--proxy-target-class为true基于cglib代理-->
<aop:aspectj-autoproxy proxy-target-class="false"/>
* 配置方式二
 @EnableAspectJAutoProxy
 public class SpringConfiguration
 * 温馨提醒：单元测试需要更改为@SpringJUnitConfig(SpringConfiguration.class)
* 代码
@Component("sa")
@Aspect
public class SpeakAspect {
    /**
     * 定义切入点
     */
    @Pointcut("execution(* com.lg.service.impl.SpeakServiceImpl.*(..))")
    public void pointcut(){
```

```
    }
    @Before("pointcut()")
    public void speakBefore(){
        System.out.println("---speakBefore---");
    }
    @Around("pointcut()")
    public Object speakAround(ProceedingJoinPoint pjp) throws Throwable {
        Object obj=null;
        System.out.println("---around前---");
        try{
            // 执行目标方法
            obj = pjp.proceed();
        }catch (Throwable e){
            System.out.println("---around异常---");
            // 监听参数为true则抛出异常，为false则捕获并不抛出异常
            if(pjp.getArgs().length>0 && (Boolean) pjp.getArgs()[0]){
                throw e;
            }else{
                obj=null;
            }
        }
        System.out.println("---around后---");
        return obj;
    }
    @After("pointcut()")
    public void speakAfter(){
        System.out.println("---speakAfter---");
    }
    @AfterReturning("pointcut()")
    public void speakAfterThrowing(){
        System.out.println("speakAfterThrowing");
    }
    @AfterThrowing("pointcut()")
    public void speakAfterReturning(){
        System.out.println("speakAfeterReturning");
    }
}

* 其他不变，执行单元测试
 * 发现环绕通知around 比 前置通知before 先执行
```

* 能够了解Spring 集成JDBC

* SpringJDBC模块是解决持久层CRUD操作，是对JDBC的一个简单封装，类型于DBUtils工具。

```
* 添加依赖
 <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.2.RELEASE</version>
 </dependency>
 <dependency>
     <groupId>mysql</groupId>
     <artifactId>mysql-connector-java</artifactId>
     <version>5.1.16</version>
 </dependency>
* 代码
 @Test
    public void test6(){
        // 1 构建JdbcTemplate模板
        JdbcTemplate jdbcTemplate=new JdbcTemplate();
        // 2 构建连接池(Spring 内置的数据库连接池)
        DriverManagerDataSource dataSource=new DriverManagerDataSource();
        dataSource.setUrl("jdbc:mysql://localhost:3306/lg01?characterEncoding=u
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
        jdbcTemplate.setDataSource(dataSource);
        String sql="SELECT id,username,psw,sex FROM USER";
        jdbcTemplate.query(sql, new RowCallbackHandler() {
            @Override
            public void processRow(ResultSet resultSet) throws SQLException {
                int id=resultSet.getInt(1);
                String username=resultSet.getString(2);
                String psw = resultSet.getString(3);
                char sex=resultSet.getString(4).charAt(0);
                System.out.println(id+":"+username+":"+psw+":"+sex);
```

```
            }
        });
    }
```