Day06

Spring 注解整合 MyBatis

分析

步骤

1. 修改 Spring 配置配置文件格式为注解格式，同时开启注解扫描。
2. 业务类使用 @Component 进行 IoC，使用 @Autowired 进行 DI
3. 建立配置文件JdbcConfig与MybatisConfig类，并将其导入到核心配置类SpringConfig
4. 使用 AnnotationConfigApplicationContext 对象加载配置项

总结

课堂作业

Resource

注解方式下配置扫描映射文件

课后作业：

AOP

概述

AspectJ

AOP 入门案例

1. 新建 maven 项目。
2. 导入依赖 spring 和 aspectJ 的依赖
3. 编写核心业务代码
4. 编写非核心业务代码
5. 通过 AOP 编程，把非核心业务代码和核心业务代码进行整合。
    1. 添加 AOP Schema
    2. 配置 bean 和 设置 aop
6. 测试

课堂作业

切入点

概念

切入点表达式

Spring 注解整合 MyBatis

分析

- 重点是不再在 xml 文件中进行 IoC 和 DI 操作
  - 业务类使用注解形式进行 IoC，属性采用注解进行 DI。
- 建立独立的配置管理类，分类管理外部资源，根据功能进行分类，并提供对应的方法获取bean
- 使用注解开启自动对 bean 所在的包进行扫描，加载所有注解配置的资源
- 使用 AnnotationConfigApplicationContext 对象加载所有的启动配置类
  - 内部使用@import注解进行进行导入和关联

步骤

1. 修改 Spring 配置配置文件格式为注解格式，同时开启注解扫描。

Java 复制代码

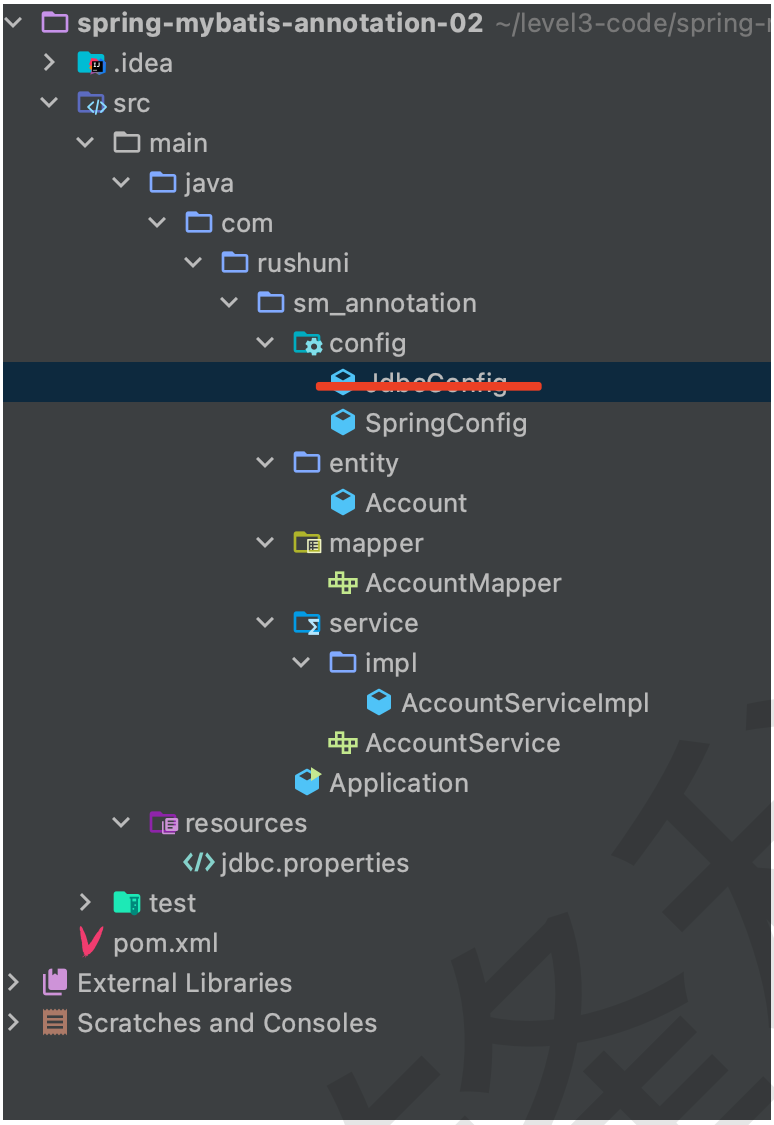
```java
@Configuration
@ComponentScan("com.rushuni")
public class SpringConfig {
}
```

2. 业务类使用 @Component 进行 IoC，使用 @Autowired 进行 DI

Java 复制代码

```java
@Service("accountService")
public class AccountServiceImpl implements AccountService {

    @Autowired
    AccountMapper accountMapper;

    // ...
}
```

上面步骤完成得到如下内容：



3. 建立配置文件JdbcConfig与MybatisConfig类，并将其导入到核心配置类SpringConfig

Java 复制代码

```java
@Configuration
@ComponentScan("com.rushuni")
@PropertySource("classpath:jdbc.properties")
@Import({JdbcConfig.class,MybatisConfig.class})
public class SpringConfig {
}
```

JdbcConfig:

Java 复制代码

```java
package com.rushuni.config;

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

import javax.sql.DataSource;

/**
 * @author rushuni
 * @date 2021年07月19日 9:07 上午
 */
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("dataSource")
    public DataSource getDataSource(){
        DruidDataSource ds = new DruidDataSource();
        ds.setDriverClassName(driver);
        ds.setUrl(url);
        ds.setUsername(userName);
        ds.setPassword(password);
        return ds;
    }
}
```

MybatisConfig

Java 复制代码

```java
package com.rushuni.config;

import org.mybatis.spring.SqlSessionFactoryBean;
import org.mybatis.spring.mapper.MapperScannerConfigurer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.Resource;

import javax.sql.DataSource;

/**
 * @author rushuni
 * @date 2021年07月19日 9:07 上午
 */
public class MybatisConfig {

    @Bean
    public SqlSessionFactoryBean getSqlSessionFactoryBean(@Autowired DataSource dataSource){
        SqlSessionFactoryBean ssfb = new SqlSessionFactoryBean();
        ssfb.setTypeAliasesPackage("com.rushuni.entity");
        ssfb.setDataSource(dataSource);
        return ssfb;
    }

    @Bean
    public MapperScannerConfigurer getMapperScannerConfigurer(){
        MapperScannerConfigurer msc = new MapperScannerConfigurer();
        msc.setBasePackage("com.rushuni.mapper");
        return msc;
    }
}
```

4. 使用 AnnotationConfigApplicationContext 对象加载配置项

Java 复制代码

```java
public class Application {

    public static void main(String[] args) {
        ApplicationContext ctx = new AnnotationConfigApplicationContext(SpringConfig.class);
        AccountService accountService = (AccountService) ctx.getBean("accountService");
        Account ac = accountService.getById(1);
        System.out.println(ac);
    }
}
```

总结


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

    <!-- 1.加载 perperties 配置文件 -->
    <context:property-placeholder location="classpath:*.properties"/>

    <!-- 2.配置 druid 数据库连接池 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!-- 3.配置 Service 层的 bean，注入该层 bean 依赖的 Mapper 对象 -->
    <bean id="accountService" class="com.rushuni.service.impl.AccountServiceImpl">
        <property name="accountMapper" ref="accountMapper"/>
    </bean>

    <!-- 4.spring 整合 mybatis 配置 -->
    <bean class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="typeAliasesPackage" value="com.rushuni.entity"/>
        <property name="mapperLocations" value="mapper/*.xml"/>
    </bean>

    <!-- 5.扫描 mapper 层，把 mapper 层的对象交给 Spring 管理，在第 3 步直接使用。-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.rushuni.mapper"/>
    </bean>

</beans>
```

课堂作业

- 完成 Spring 注解整合 MyBatis 项目。

Resource

Prefix	Example	Explanation
classpath:	classpath:com/myapp/config.xml	Loaded from the classpath.
file:	file:///data/config.xml	Loaded as a URL from the filesystem. See also FileSystemResourceCaveats.
https:	https://myserver/logo.png	Loaded as a URL.
(none)	/data/config.xml	Depends on the underlying ApplicationContext.

注解方式下配置扫描映射文件

关键代码：

```java
final PathMatchingResourcePatternResolver pathMatchingResourcePatternResolver = new PathMatchingResourcePatternResolver();
ssfb.setMapperLocations(pathMatchingResourcePatternResolver.getResources("mapper/*.xml"));

```

课后作业：

- 修改为 MyBatis 使用 xml 的方式进行开发。

AOP

概述

在 AOP 思想中，功能主要有两点

- 核心功能（核心业务功能，如对数据的增删改查等）
- 辅助功能（非核心业务功能，如日志记录，性能统计等等）

核心功能因此也称作切点，辅助功能因此成为切面。

简单来说，在开发的时候，将这两点分别进行独立开发，然后进行这两点的整合，这就叫 AOP编程。

AspectJ

Aspect（切面）用于描述切入点与通知间的关系，是AOP编程中的一个概念。

AspectJ 是基于 java 语言对 Aspect 的实现。

AOP 入门案例

1. 新建 maven 项目。

2. 导入依赖 spring 和 aspectJ 的依赖

XML 复制代码

```xml
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.3.9</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>5.3.9</version>
</dependency>
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.9.7</version>
</dependency>
```

3. 编写核心业务代码

创建 service 包，UserService 类

4. 编写非核心业务代码

创建 aop 包，AopAdvice 类

5. 通过 AOP 编程，把非核心业务代码和核心业务代码进行整合。

1. 添加 AOP Schema

XML 复制代码

```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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- bean definitions here -->

</beans>
```

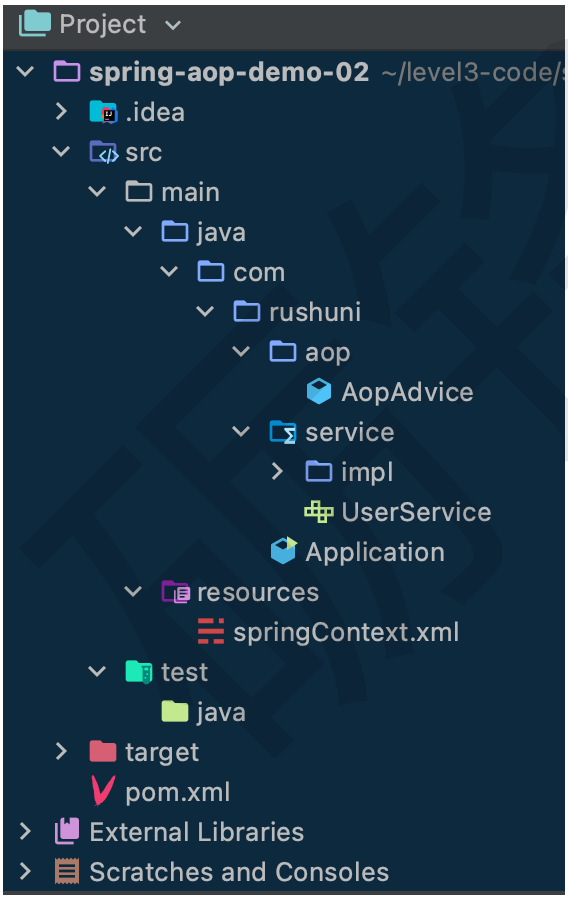
2. 配置 bean 和 设置 aop

XML 复制代码

```xml
<!-- bean definitions here -->
<bean id="userService" class="com.rushuni.service.impl.UserServiceImpl">
</bean>
<bean id="aopAdvice" class="com.rushuni.aop.AopAdvice"></bean>

<!-- AOP 设置 -->
<aop:config>
  <!-- 配置切点（核心业务类）-->
  <aop:pointcut id="pointCut1"
                expression="execution(*
com.rushuni.service.impl.UserServiceImpl.*(..))"/>
  <!-- 配置切面（要添加到核心业务中执行的类） -->
  <aop:aspect id="myAspect" ref="aopAdvice">
    <!-- 执行的方法，before：在之前执行，也就是在 UserServiceImpl 所有方法执行前执行
printTime 方法 -->
    <aop:before method="printTime" pointcut-ref="pointCut1"/>
  </aop:aspect>
</aop:config>
```

6. 测试

```java
package com.rushuni;

import com.rushuni.service.UserService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author rushuni
 * @date 2021年07月19日 3:59 下午
 */
public class Application {
    public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("springContext.xml");
        final UserService userService = (UserService)ctx.getBean("userService");
        userService.save();
        userService.delete();
    }
}
```



课堂作业

- 完成 AOP 入门案例，熟悉切点和切面。

切入点

概念

- 切入点描述的是某个方法
- 切入点表达式是一个快速匹配方法描述的通配格式，类似于正则表达式

切入点表达式

组成

```XML
关键字（访问修饰符  返回值  包名.类名.方法名（参数）异常名）
```

说明：

- 关键字：描述表达式的匹配模式（参看关键字列表）
- 访问修饰符：方法的访问控制权限修饰符
- 类名：方法所在的类（此处可以配置接口名称）
- 异常：方法定义中指定抛出的异常

范例

```XML
execution (public User com.rushuni.service.UserService.getById(int))
```

关键字

- execution ： 匹配执行指定方法
- args ： 匹配带有指定参数类型的方法

其他的（自行翻译）：

Spring AOP supports the following AspectJ pointcut designators (PCD) for use in pointcut expressions:

- execution: For matching method execution join points. This is the primary pointcut designator to use when working with Spring AOP.
- within: Limits matching to join points within certain types (the execution of a method declared within a matching type when using Spring AOP).

- this: Limits matching to join points (the execution of methods when using Spring AOP) where the bean reference (Spring AOP proxy) is an instance of the given type.
- target: Limits matching to join points (the execution of methods when using Spring AOP) where the target object (application object being proxied) is an instance of the given type.
- args: Limits matching to join points (the execution of methods when using Spring AOP) where the arguments are instances of the given types.
- @target: Limits matching to join points (the execution of methods when using Spring AOP) where the class of the executing object has an annotation of the given type.
- @args: Limits matching to join points (the execution of methods when using Spring AOP) where the runtime type of the actual arguments passed have annotations of the given types.
- @within: Limits matching to join points within types that have the given annotation (the execution of methods declared in types with the given annotation when using Spring AOP).
- @annotation: Limits matching to join points where the subject of the join point (the method being run in Spring AOP) has the given annotation.

通配符

- *：单个独立的任意符号，可以独立出现，也可以作为前缀或者后缀的匹配符出现

```XML
execution (public * com.rushuni.*.UserService.get*(*))
```

匹配 com.rushuni 包下的任意包中的 UserService 类或接口中所有 get 开头的带有一个参数的方法

- .. ：多个连续的任意符号，可以独立出现，常用于简化包名与参数的书写

```XML
execution (public User com..UserService.getById(..))
```

匹配 com 包下的任意包中的 UserService 类或接口中所有名称为 getById 的方法

- +：专用于匹配子类类型

```XML
execution(* *..*Service+.*(..))
```

逻辑运算符

- &&： 连接两个切入点表达式，表示两个切入点表达式同时成立的匹配
- ||： 连接两个切入点表达式，表示两个切入点表达式成立任意一个的匹配
- !： 连接单个切入点表达式，表示该切入点表达式不成立的匹配

例子

XML 复制代码

```xml
execution(* *(..))
execution(* *..*(..))
execution(* *..*.*(..))
execution(public * *..*.*(..))
execution(public int *..*.*(..))
execution(public void *..*.*(..))
execution(public void com..*.*(..))
execution(public void com..service.*.*(..))
execution(public void com.rushuni.service.*.*(..))
execution(public void com.rushuni.service.User*.*(..))
execution(public void com.rushuni.service.*Service.*(..))
execution(public void com.rushuni.service.UserService.*(..))
execution(public User com.rushuni.service.UserService.find*(..))
execution(public User com.rushuni.service.UserService.*Id(..))
execution(public User com.rushuni.service.UserService.findById(..))
execution(public User com.rushuni.service.UserService.findById(int))
execution(public User com.rushuni.service.UserService.findById(int,int))
execution(public User com.rushuni.service.UserService.findById(int,*))
execution(public User com.rushuni.service.UserService.findById(*,int))
execution(public User com.rushuni.service.UserService.findById())
execution(List com.rushuni.service.*Service+.findAll(..))
```

AOP 配置（XML）

AOP 配置在 XML 中主要有三个标签：aop:config、aop:aspect、aop:pointcut。

aop:config

在 beans 标签中可以使用 aop:config 标签用于开始一个 AOP 的设置。

格式

```xml
<beans>
  <aop:config>……</aop:config>
  <aop:config>……</aop:config>
</beans>
```

说明

一个beans标签中可以配置多个aop:config标签

aop:aspect

在 aop:config 标签中可以使用 aop:aspect 设置具体的 AOP 通知对应的切入点

格式

```xml
<aop:config>
  <aop:aspect ref="beanId">……</aop:aspect>
  <aop:aspect ref="beanId">……</aop:aspect>
</aop:config>
```

说明

一个 aop:config 标签中可以配置多个 aop:aspect 标签

基本属性

ref ： 通知所在的 bean 的 id

aop:pointcut

aop:pointcut 标签用于设置切入点，可以在 aop:config 标签中设置，也可以在 aop:aspect 标签中设置。

格式

```xml
<aop:config>
  <aop:pointcut id="pointcutId" expression="……"/>
  <aop:aspect>
    <aop:pointcut id="pointcutId" expression="……"/>
  </aop:aspect>
</aop:config>
```

说明

一个 aop:config 标签中可以配置多个 aop:pointcut 标签，且该标签可以配置在 aop:aspect 标签内。

基本属性

id ： 识别切入点的名称

expression ： 切入点表达式

切入点的几种配置

```xml
<aop:config>
  <!--配置公共切入点-->
  <aop:pointcut id="pt1" expression="execution(* *(..))"/>
  <aop:aspect ref="myAdvice">
    <!--配置局部切入点-->
    <aop:pointcut id="pt2" expression="execution(* *(..))"/>
    <!--引用公共切入点-->
    <aop:before method="logAdvice" pointcut-ref="pt1"/>
    <!--引用局部切入点-->
    <aop:before method="logAdvice" pointcut-ref="pt2"/>
    <!--直接配置切入点-->
    <aop:before method="logAdvice" pointcut="execution(* *(..))"/>
  </aop:aspect>
</aop:config>
```

AOP 的通知类型

概述

AOP 通知类型主要有五种：

- 前置通知：原始方法执行前执行，如果通知中抛出异常，阻止原始方法运行

    - 应用：数据校验
- 后置通知：原始方法执行后执行，无论原始方法中是否出现异常，都将执行通知
    - 应用：现场清理
- 返回后通知：原始方法正常执行完毕并返回结果后执行，如果原始方法中抛出异常，无法执行
    - 应用：返回值相关数据处理
- 抛出异常后通知：原始方法抛出异常后执行，如果原始方法没有抛出异常，无法执行
    - 应用：对原始方法中出现的异常信息进行处理
- 环绕通知：在原始方法执行前后均有对应执行执行，还可以阻止原始方法的执行
    - 应用：十分强大，可以做任何事情

前置通知（aop:before）

在 aop:aspec 标签中可以设置前置通知

XML 复制代码

```xml
<aop:aspect ref="adviceId">
  <aop:before method="methodName" pointcut="……"/>
</aop:aspect>
```

- 说明：一个 aop:aspect 标签中可以配置多个 aop:before 标签
- 基本属性：
    - method ： 在通知类中设置当前通知类别对应的方法
    - pointcut ： 设置当前通知对应的切入点表达式，与 pointcut-ref 属性冲突
    - pointcut-ref ： 设置当前通知对应的切入点id，与 pointcut 属性冲突

后置通知（aop:after）

在 aop:aspect 标签中可以设置后置通知

XML 复制代码

```xml
<aop:aspect ref="adviceId">
  <aop:after method="methodName" pointcut="……"/>
</aop:aspect>
```

- 说明：一个 aop:aspect 标签中可以配置多个 aop:after 标签
- 基本属性：

- method ： 在通知类中设置当前通知类别对应的方法
- pointcut ： 设置当前通知对应的切入点表达式，与pointcut-ref属性冲突
- pointcut-ref ： 设置当前通知对应的切入点id，与pointcut属性冲突

返回后通知（aop:after-returning）

在 aop:aspect 标签中可以设置返回后通知

XML 复制代码

```xml
<aop:aspect ref="adviceId">
  <aop:after-returning method="methodName" pointcut="……"/>
</aop:aspect>
```

- 说明：一个aop:aspect标签中可以配置多个aop:after-returning标签
- 基本属性：
  - method ： 在通知类中设置当前通知类别对应的方法
  - pointcut ： 设置当前通知对应的切入点表达式，与 pointcut-ref 属性冲突
  - pointcut-ref ： 设置当前通知对应的切入点id，与 pointcut 属性冲突

抛出异常后通知（aop:after-throwing）

在 aop:aspect 标签中可以设置抛出异常后通知

XML 复制代码

```xml
<aop:aspect ref="adviceId">
  <aop:after-throwing method="methodName" pointcut="……"/>
</aop:aspect>
```

- 说明：一个 aop:aspect 标签中可以配置多个 aop:after-throwing 标签
- 基本属性：
- method ： 在通知类中设置当前通知类别对应的方法
- pointcut ： 设置当前通知对应的切入点表达式，与pointcut-ref属性冲突
- pointcut-ref ： 设置当前通知对应的切入点id，与pointcut属性冲突

环绕通知（aop:around）

在 aop:aspect 标签中可以设置环绕通知

```xml
<aop:aspect ref="adviceId">
  <aop:around method="methodName" pointcut="……"/>
</aop:aspect>
```

- 说明：一个aop:aspect标签中可以配置多个aop:around标签
- 基本属性：
  - method ： 在通知类中设置当前通知类别对应的方法
  - pointcut ： 设置当前通知对应的切入点表达式，与pointcut-ref属性冲突
  - pointcut-ref ： 设置当前通知对应的切入点id，与pointcut属性冲突

环绕通知的开发方式

环绕通知是在原始方法的前后添加功能，在环绕通知中，存在对原始方法的显式调用

```java
public Object around(ProceedingJoinPoint pjp) throws Throwable {
    Object ret = pjp.proceed();
    return ret;
}
```

- 环绕通知方法相关说明：
  - 方法须设定Object类型的返回值，否则会拦截原始方法的返回。
  - 如果原始方法返回值类型为void，通知方也可以设定返回值类型为void，最终返回null。
  - 方法需在第一个参数位置设定 **ProceedingJoinPoint** 对象，通过该对象调用 proceed() 方法，实现对原始方法的调用。如省略该参数，原始方法将无法执行。
  - 使用 proceed() 方法调用原始方法时，因无法预知原始方法运行过程中是否会出现异常，强制抛出Throwable对象，封装原始方法中可能出现的异常信息。

通知顺序

当同一个切入点配置了多个通知时，通知会存在运行的先后顺序，该顺序以通知配置的顺序为准。

课后作业

- 自行完成通过注解整合 Spring 和 MybBatis，注意 Mybatis 有注解和 xml 两种方式。
- 自行完成 AOP 入门案例，熟悉切入点表达式，完成五种通知。