

上节课的回顾

- 什么是 控制反转 loc
- 什么是 自动装配和自动注入 DI
 - 自动注入的几种方式
 - set 注入
 - 构造器注入
 - 命名空间注入
- 使用 配置XML的方式 完成一个 基础 MVC 的搭建
 - 使用 loc 完成 Bean对象的创建
 - 使用 DI 完成 Bean 对象的依赖配置

学习注解配置loc

- 什么是 注解
- 注解配置 loc 共分为几个步骤
 - 引入依赖
 - 创建配置文件
 - 开启扫描 //分为注解和配置类
 - 依赖注入
- Spring loc 用来创建Bean对象的注解
 - @Component #这个注解的意思是 配置一个Bean对象
 - @Repository #这个注解的意思是 配置一个存储类型的 Bean对象
 - @Service #这个注解的意思是 配置一个业务类型的 Bean对象
 - @Controlled #这个注解的意思是 配置一个 控制器类型的 Bean对象
- AutoWired
 - 属性注入 # 这个是使用最多的 注入方法， 只需要在属性上添加 本注解 就可以自动的匹配类型进行注入
 - set注入 #这个注入方法需要我们创建 set注入方法才能够使用， 不推荐
 - 构造方法注入 #在有些Bean对象使用的时候， 可能存在我们需要自定义构造器的时候， 会使用此方法
 - 形参注入 #在 Mybatis 接口上面最常见， 定义的查询方法， SQL参数都是通过 形参注入进去的
 - 使用 @Qualifier 修改依赖类型
- @Resource
 - Java JDK的标准注解， AutoWired 是Spring框架的注解
 - AutoWired 默认使用ByType装配， AutoWired 可以用在属性， set方法， 构造器， 构造器参数
 - Resource 根据Byname， name找不到会切换ByType 可以用在属性和set方法上，
- 抛弃Spring 配置档
 - 创建配置类
 - Configuration 声明配置类

AOP 需求分析

- 核心业务 和 非核心业务的分离
- Spring 框架层 大量使用了AOP技术
- 使用代理对象 曲线救国
 - 静态代理
 - 动态代理
- AOP 实例的模拟
- 动态代理实现
 - 有接口的情况 JDK 动态代理 实现接口
 - 没有接口的情况 cglib 动态代理 继承对象
 - 面试可能被问的题目 Spring Aop实现原理
- Spring AOP
 - Spring aop 名词
 - 切入点 面向切片变成，切片
 - 通知：
 - 前置通知：在被代理的目标方法**前**执行
 - 返回通知：在被代理的目标方法**成功结束**后执行
 - 后置通知：在被代理的目标方法**最终结束**后执行
 - 异常通知：在被代理的目标方法**异常结束**后执行
 - 环绕通知：包括上面四种通知对应的所有位置
 - 通知流程： 前置-调用-返回-后置
 - 切面：就是接收通知的类 叫切面类， 切面类封装了通知
 - 目标：被代理的对象
 - 代理：创建的代理目标的对象， 叫代理
 - 连接点：（概念）就是通知出发的点
 - 切入点：就是要代理的方法
 - 切入点表达式
 - aspects AOP框架 spring 使用的aop代理框架
 - 使用流程
 - 开启 aspects 功能
 - 配置切面类
 - 配置切入点 并 配置表达式
 - 配置通知方法

1. 课程XML配置

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

```
2 <beans xmlns="
  http://www.springframework.org/schema/beans
  <beans xmlns="
3     xmlns:xsi="
  http://www.w3.org/2001/XMLSchema-instance
  xmlns:xsi="
4     xmlns:context="
  http://www.springframework.org/schema/context
  xmlns:context="
5     xmlns:aop="
  http://www.springframework.org/schema/aop
  xmlns:aop="
6     xsi:schemaLocation="
7
  http://www.springframework.org/schema/aop

  http://www.springframework.org/schema/aop/spring-aop.xsd

8
  http://www.springframework.org/schema/beans

  http://www.springframework.org/schema/beans/spring-beans.xsd

9
  http://www.springframework.org/schema/context

  http://www.springframework.org/schema/context/spring-context.xsd

10
11 ">
12 <context:component-scan base-package="com.lovecoding">
13 </context:component-scan>
14 <!-- 开启AOP自动代理 -->
15     <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
16     <aop:config>
17         <aop:pointcut id="counter" expression="execution(*
  com.lovecoding.aop.CountterImpl.*(..))" />
18         <aop:aspect ref="counterPorxy">
19             <aop:after method="AfterAdd" pointcut-ref="counter"></aop:after>
20             <aop:before method="add" pointcut-ref="counter" ></aop:before>
21             <aop:after-returning method="afterReturning" pointcut-ref="counter" >
  </aop:after-returning>
22             <aop:around method="around" pointcut-ref="counter" />
23             <aop:after-throwing method="AfterThrowing" pointcut-ref="counter" />
24         </aop:aspect>
```

```
25     </aop:config>
```

```
26 </beans>
```

2. AOP切片类

```
1  package com.lovecoding.aop;
2  import org.aspectj.lang.JoinPoint;
3  import org.aspectj.lang.ProceedingJoinPoint;
4  import org.aspectj.lang.Signature;
5  import org.aspectj.lang.annotation.*;
6  import org.springframework.stereotype.Component;
7  //@Aspect
8  @Component
9  public class CounterPorxy {
10
11      /**
12       * 接收通知的方法
13       * Before 前置通知
14       * After 后置通知
15       * AfterReturning 返回通知
16       * Around 环绕通知
17       * AfterThrowing 异常通知
18       */
19      @Before("execution(* com.lovecoding.aop.CounttterImpl.*(..))")
20      public void add(JoinPoint joinPoint){
21          System.out.println( "前置通知" );
22      }
23
24      @After("execution(public int com.lovecoding.aop.CounttterImpl.*(..))")
25      public void AfterAdd(JoinPoint joinPoint){
26          Signature signature = joinPoint.getSignature();
27          String name = signature.getName();
28          //System.out.println( "切入点 方法" + name );
29          System.out.println( "后置通知" );
30      }
31
32      @AfterReturning("execution(public int com.lovecoding.aop.CounttterImpl.*(..))")
33      public void afterReturning(JoinPoint joinPoint){
34          System.out.println( "返回通知" );
```

```
35     }
36
37     @Around("execution(public int com.lovecoding.aop.CountterImpl.*(..))")
38     public Object around( ProceedingJoinPoint joinPoint ){
39         try {
40             System.out.println( "环绕通知" );
41             return joinPoint.proceed(joinPoint.getArgs());
42         } catch (Throwable e) {
43             e.printStackTrace();
44         }
45         return null;
46     }
47
48     @AfterThrowing("execution(public int com.lovecoding.aop.CountterImpl.*(..))")
49     public void AfterThrowing(JoinPoint joinPoint){
50         Signature signature = joinPoint.getSignature();
51         String name = signature.getName();
52         System.out.println( "方法 : " + name + "发生异常" );
53     }
54 }
55
```

3.