

SpringAop的认识

01、课程目标

- 认识SpringAop
- 了解SpringAop的底层实现的原理
 - JDK动态代理
 - CGLIB代理
- SpringAop的增强通知的类型
- SpringAop的切点的定义
- Spring中默认的Aop代理机制是什么？
- SpringBoot中默认的Aop代理机制是什么？
- SpringAop的实战开发，日志管理，限流处理，权限拦截。
- SpringMvc源码分析后置通知是如何和Aop产生管理的。
- 然后在回归学习SpringAop和动态代理的关系。
- 为什么springaop是构建在ioc基础上的呢？

02、准备工作

- 创建一个springboot-aop-07的项目
- 在项目的pom.xml中依赖aop如下：

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-aop</artifactId>
4 </dependency>
```

03、认识SpringAop

java学习曲线，面向对象，创建类，创建属性，创建方式，创建对象，执行方法，给属性赋值，比如：

用户pojo

```
1 package com.kuangstudy.first;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
7
8 /**
9  * @author 飞哥
10  * @Title: 学相伴出品
11  * @Description: 飞哥B站地址:
12  * https://space.bilibili.com/490711252
13  * 记得关注和三连哦!
14  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
15  * @date 2021/12/21 22:05
16  */
17 @Data
18 @NoArgsConstructor
19 @AllArgsConstructor
20 @ToString
21 public class User {
22     private Integer userId;
23     private String nickname;
24     private String password;
25 }
```

订单pojo

```

1 package com.kuangstudy.first;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
7
8 /**
9  * @author 飞哥
10  * @Title: 学相伴出品
11  * @Description: 飞哥B站地址:
12  * https://space.bilibili.com/490711252
13  * 记得关注和三连哦!
14  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
15  * @date 2021/12/21 22:05
16  */
17 @Data
18 @NoArgsConstructor
19 @AllArgsConstructor
20 @ToString
21 public class Order {
22     private Integer userId;
23     private Integer orderId;
24     private String price;
25     private String orderTradeNo;
26 }

```

用户service

```

1 package com.kuangstudy.first;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
7 import lombok.extern.slf4j.Slf4j;

```

```
8  import org.springframework.stereotype.Service;
9
10 /**
11  * @author 飞哥
12  * @Title: 学相伴出品
13  * @Description: 飞哥B站地址:
14  * https://space.bilibili.com/490711252
15  * 记得关注和三连哦!
16  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
17  * @date 2021/12/21 22:05
18  */
19 @Slf4j
20 @Service
21 public class UserService {
22     /**
23     * 用户注册
24     *
25     * @param user
26     */
27     public void saveUser(User user) {
28         log.info("用户注册....");
29     }
30
31     /**
32     * 修改用户
33     *
34     * @param user
35     */
36     public void updateUser(User user) {
37         log.info("修改用户....");
38     }
39
40     /**
41     * 删除用户
42     *
43     * @param userId
```

```

45     */
46     public void delUser(Integer userId) {
47         log.info("删除用户....{}", userId);
48     }
49
50 }
51

```

订单service

```

1  package com.kuangstudy.first;
2
3  import lombok.extern.slf4j.Slf4j;
4  import org.springframework.stereotype.Service;
5
6  /**
7   * @author 飞哥
8   * @Title: 学相伴出品
9   * @Description: 飞哥B站地址:
10    https://space.bilibili.com/490711252
11   * 记得关注和三连哦!
12   * @Description: 我们有一个学习网站: https://www.kuangstudy.com
13   * @date 2021/12/21 22:05
14   */
15  @Slf4j
16  @Service
17  public class OrderService {
18
19      /**
20       * 订单注册
21       *
22       * @param order
23       */
24      public void saveOrder(Order order) {
25          log.info("订单注册....");
26      }
27
28      /**
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

28      * 修改订单
29      *
30      * @param order
31      */
32      public void updateOrder(Order order) {
33          log.info("修改订单....");
34      }
35
36
37      /**
38      * 删除订单
39      *
40      * @param orderId
41      */
42      public void delOrder(Integer orderId) {
43          log.info("删除订单....{}", orderId);
44      }
45
46  }
47

```

测试代码

```

1  package com.kuangstudy;
2
3  import com.kuangstudy.first.User;
4  import com.kuangstudy.first.UserService;
5  import org.junit.jupiter.api.Test;
6  import
    org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.boot.test.context.SpringBootTest;
8
9  @SpringBootTest
10 class SpringbootAopApplicationTests {
11
12     @Autowired

```

```

13     private UserService userService;
14
15
16     @Test
17     void contextLoads() {
18         // 1: 用户注册
19         userService.saveUser(new User());
20         // 2: 用户修改
21         userService.updateUser(new User());
22         // 3: 用户的删除
23         userService.delUser(1);
24     }
25
26 }
27

```

打印测试结果如下：

```

1  【KSD - CONSOLE】 2021-12-21 22:15:13:275 [main] [INFO ]
   com.kuangstudy.first.UserService saveUser 28 - 用户注册....
2  【KSD - CONSOLE】 2021-12-21 22:15:13:275 [main] [INFO ]
   com.kuangstudy.first.UserService updateUser 37 - 修改用户....
3  【KSD - CONSOLE】 2021-12-21 22:15:13:275 [main] [INFO ]
   com.kuangstudy.first.UserService delUser 47 - 删除用户....1

```

上面代码：其实就一个标准的java面向对象，创建对象执行方法的过程。如果在开发过程中，我们需要给对象执行方法方法的时候，去增强方法的执行逻辑的时候，控制权限的时候，进行限流的时候，怎么办呢？

重要的信息

对象执行方法 = ,aop它其实就是对象执行方法过程中一种横切逻辑增强处理的机制。-- OOP

我们写代码追去的境界

- 高内聚低耦合
 - 接口
 - oop
 - 消息队列
 - 设计模式
 - SPI
- 尽量用面向对象的思维
 - 枚举

04、日志开发需求场景

我们现在要给程序中的所有用户接口和订单的接口全部进行订单增强日志处理？

会定义一个 Log 日志类：

日志场景需求的作用

- 可以协助我们排除执行方法耗时的问题
- 这样可以针对性的优化和处理执行方法耗时的问题

日志的pojo

```
1 package com.kuangstudy.first;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
```



```

7
8 /**
9  * @author 飞哥
10 * @Title: 学相伴出品
11 * @Description: 飞哥B站地址:
    https://space.bilibili.com/490711252
12 * 记得关注和三连哦!
13 * @Description: 我们有一个学习网站: https://www.kuangstudy.com
14 * @date 2021/12/21 22:05
15 */
16 @Data
17 @NoArgsConstructor
18 @AllArgsConstructor
19 @ToString
20 public class Logs {
21     private Integer id;
22     private String classname;
23     private String method;
24     private String time;
25     private String params;
26 }
27

```

日志处理service

```

1 package com.kuangstudy.first;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.springframework.stereotype.Service;
5
6 /**
7  * @author 飞哥
8  * @Title: 学相伴出品
9  * @Description: 飞哥B站地址:
    https://space.bilibili.com/490711252
10 * 记得关注和三连哦!
11 * @Description: 我们有一个学习网站: https://www.kuangstudy.com

```

```

12  * @date 2021/12/21 22:05
13  */
14  @Slf4j
15  @Service
16  public class LogService {
17
18      public void saveLog(Logs logs) {
19          log.info("你保存日志是: {}", logs);
20      }
21  }
22

```

开发日志保存的记录功能

我们一般的正常思维和开发流程是上面样子，如下：

用户服务需要记录日志如下：

```

1  package com.kuangstudy.first;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6  import lombok.ToString;
7  import lombok.extern.slf4j.Slf4j;
8  import
    org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.stereotype.Service;
10
11  /**
12   * @author 飞哥
13   * @Title: 学相伴出品
14   * @Description: 飞哥B站地址:
    https://space.bilibili.com/490711252

```

```
15  * 记得关注和三连哦！
16  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
17  * @date 2021/12/21 22:05
18  */
19 @Slf4j
20 @Service
21 public class UserService {
22
23
24     @Autowired
25     private LogService logService;
26
27     /**
28      * 用户注册
29      *
30      * @param user
31      */
32     public void saveUser(User user) {
33         log.info("用户注册....");
34
35         // 记录日志
36         Logs logs = new Logs();
37
38         logs.setClassname("com.kuangstudy.service.UserService");
39         logs.setMethod("saveUser");
40         logs.setId(1);
41         logs.setParams(user.toString());
42         logs.setTime("100ms");
43         logService.saveLog(logs);
44     }
45
46     /**
47      * 修改用户
48      *
49      * @param user
50      */
51     public void updateUser(User user) {
52         log.info("修改用户....");
53     }
54 }
```

```
52
53     // 记录日志
54     Logs logs = new Logs();
55
56     logs.setClassname("com.kuangstudy.service.UserService");
57     logs.setMethod("updateUser");
58     logs.setId(1);
59     logs.setParams(user.toString());
60     logs.setTime("100ms");
61     logService.saveLog(logs);
62 }
63
64 /**
65  * 删除用户
66  *
67  * @param userId
68  */
69 public void delUser(Integer userId) {
70     log.info("删除用户....{}", userId);
71
72     // 记录日志
73     Logs logs = new Logs();
74
75     logs.setClassname("com.kuangstudy.service.UserService");
76     logs.setMethod("delUser");
77     logs.setId(1);
78     logs.setParams(userId + "");
79     logs.setTime("100ms");
80     logService.saveLog(logs);
81 }
82 }
83
```

```
1 package com.kuangstudy.first;
2
3 import lombok.extern.slf4j.Slf4j;
4 import
    org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 /**
8  * @author 飞哥
9  * @Title: 学相伴出品
10  * @Description: 飞哥B站地址:
    https://space.bilibili.com/490711252
11  * 记得关注和三连哦!
12  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
13  * @date 2021/12/21 22:05
14  */
15 @Slf4j
16 @Service
17 public class OrderService {
18
19     @Autowired
20     private LogService logService;
21
22     /**
23      * 订单注册
24      *
25      * @param order
26      */
27     public void saveOrder(Order order) {
28         log.info("订单注册....");
29
30         // 记录日志
31         Logs logs = new Logs();
32
33         logs.setClassname("com.kuangstudy.service.OrderService");
34         logs.setMethod("saveOrder");
```

```
34         logs.setId(1);
35         logs.setParams(order.toString());
36         logs.setTime("100ms");
37         logService.saveLog(logs);
38     }
39
40     /**
41      * 修改订单
42      *
43      * @param order
44      */
45     public void updateOrder(Order order) {
46         log.info("修改订单....");
47
48         // 记录日志
49         Logs logs = new Logs();
50
51         logs.setClassname("com.kuangstudy.service.OrderService");
52         logs.setMethod("updateOrder");
53         logs.setId(1);
54         logs.setParams(order.toString());
55         logs.setTime("100ms");
56         logService.saveLog(logs);
57     }
58
59     /**
60      * 删除订单
61      *
62      * @param orderId
63      */
64     public void delOrder(Integer orderId) {
65         log.info("删除订单....{}", orderId);
66
67         // 记录日志
68         Logs logs = new Logs();
69
70         logs.setClassname("com.kuangstudy.service.OrderService");
```

```

70         logs.setMethod("delOrder");
71         logs.setId(1);
72         logs.setParams(orderId.toString());
73         logs.setTime("100ms");
74         logService.saveLog(logs);
75     }
76
77 }
78

```

执行测试

```

1  package com.kuangstudy;
2
3  import com.kuangstudy.first.Order;
4  import com.kuangstudy.first.OrderService;
5  import com.kuangstudy.first.User;
6  import com.kuangstudy.first.UserService;
7  import org.junit.jupiter.api.Test;
8  import
    org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.boot.test.context.SpringBootTest;
10
11  @SpringBootTest
12  public class SpringbootAopApplicationTests {
13
14      @Autowired
15      private UserService userService;
16      @Autowired
17      private OrderService orderService;
18
19
20      @Test
21      void contextLoads() {
22          // 1: 用户注册
23          userService.saveUser(new User());

```

```
24      // 2: 用户修改
25      userService.updateUser(new User());
26      // 3: 用户的删除
27      userService.delUser(1);
28  }
29
30  @Test
31  public void testOrder() {
32      // 1: 用户注册
33      orderService.saveOrder(new Order());
34      // 2: 用户修改
35      orderService.updateOrder(new Order());
36      // 3: 用户的删除
37      orderService.delOrder(1);
38  }
39
40  }
41
```

 执行结果


```
1  【KSD - CONSOLE】 2021-12-21 22:33:38:208 [main] [INFO ]
   com.kuangstudy.first.UserService saveUser 33 - 用户注册....
2  【KSD - CONSOLE】 2021-12-21 22:33:38:208 [main] [INFO ]
   com.kuangstudy.first.LogService saveLog 19 - 你保存日志是:
   Logs(id=1, classname=com.kuangstudy.service.UserService,
   method=saveUser, time=100ms, params=User(userId=null,
   nickname=null, password=null))
3  【KSD - CONSOLE】 2021-12-21 22:33:38:210 [main] [INFO ]
   com.kuangstudy.first.UserService updateUser 51 - 修改用户....
4  【KSD - CONSOLE】 2021-12-21 22:33:38:211 [main] [INFO ]
   com.kuangstudy.first.LogService saveLog 19 - 你保存日志是:
   Logs(id=1, classname=com.kuangstudy.service.UserService,
   method=updateUser, time=100ms, params=User(userId=null,
   nickname=null, password=null))
5  【KSD - CONSOLE】 2021-12-21 22:33:38:211 [main] [INFO ]
   com.kuangstudy.first.UserService delUser 70 - 删除用户....1
6  【KSD - CONSOLE】 2021-12-21 22:33:38:211 [main] [INFO ]
   com.kuangstudy.first.LogService saveLog 19 - 你保存日志是:
   Logs(id=1, classname=com.kuangstudy.service.UserService,
   method=delUser, time=100ms, params=1)
7
8
```

 订单执行的结果

```
1  【KSD - CONSOLE】 2021-12-21 22:34:38:089 [main] [INFO ]
   com.kuangstudy.first.OrderService saveOrder 28 - 订单注册....
2  【KSD - CONSOLE】 2021-12-21 22:34:38:090 [main] [INFO ]
   com.kuangstudy.first.LogService saveLog 19 - 你保存日志是:
   Logs(id=1, classname=com.kuangstudy.service.OrderService,
   method=saveOrder, time=100ms, params=Order(userId=null,
   orderId=null, price=null, orderTradeNo=null))
3  【KSD - CONSOLE】 2021-12-21 22:34:38:096 [main] [INFO ]
   com.kuangstudy.first.OrderService updateOrder 46 - 修改订单....
4  【KSD - CONSOLE】 2021-12-21 22:34:38:096 [main] [INFO ]
   com.kuangstudy.first.LogService saveLog 19 - 你保存日志是:
   Logs(id=1, classname=com.kuangstudy.service.OrderService,
   method=updateOrder, time=100ms, params=Order(userId=null,
   orderId=null, price=null, orderTradeNo=null))
5  【KSD - CONSOLE】 2021-12-21 22:34:38:097 [main] [INFO ]
   com.kuangstudy.first.OrderService delOrder 65 - 删除订单....1
6  【KSD - CONSOLE】 2021-12-21 22:34:38:097 [main] [INFO ]
   com.kuangstudy.first.LogService saveLog 19 - 你保存日志是:
   Logs(id=1, classname=com.kuangstudy.service.OrderService,
   method=delOrder, time=100ms, params=1)
7
8
```

🤖 你必须从上面的代码中认知又哪些？

- 1、通过上面一个非常典型案例就给一些做日志增强处理的时候，我们发现都是 **对象执行方法** 要 **额外做一些事情** 而这些个事情，你又还得不得不做。
- 2、日志逻辑的代码是紧耦合，它污染了一些业务逻辑。而且不便于后续的维护。
- 3、现在上面的整个日志都会拦截，灵活吗？如果我要考虑到有些方法要拦截，又方法又不拦截怎么处理？比如后续的限流，权限拦截

05、上面的问题的解决方案

- AOP

06、上面的代码我们可以通过一些方案去优化呢？

01、方法封装

01、日志方法的重载

```
1 package com.kuangstudy.first;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.springframework.stereotype.Service;
5
6 /**
7  * @author 飞哥
8  * @Title: 学相伴出品
9  * @Description: 飞哥B站地址:
10  * https://space.bilibili.com/490711252
11  * 记得关注和三连哦！
12  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
13  * @date 2021/12/21 22:05
14  */
15 @Slf4j
16 @Service
17 public class LogService {
18
19     public void saveLog(Logs logs) {
20         log.info("你保存日志是: {}", logs);
21     }
22 }
```

```

22     public void saveLog(String classname,String method,String
time,String params) {
23         Logs logs = new Logs();
24
25         logs.setClassname("com.kuangstudy.service.OrderService");
26         logs.setMethod("updateOrder");
27         logs.setId(1);
28         logs.setParams(params);
29         logs.setTime("100ms");
30         log.info("你保存日志是: {}", logs);
31     }
32 }

```

用户注册方法进行修改

```

1
2     /**
3     * 用户注册
4     *
5     * @param user
6     */
7     public void saveUser(User user) {
8         log.info("用户注册....");
9         // 记录日志
10
11         logService.saveLog("com.kuangstudy.service.UserService","saveUser","100ms",user.toString());
12     }

```

其他的以此类推

02: 问题

并没用解决根本性的问题，这个肯定抛弃的。

拦截器(springmvc的拦截器)

核心思想：jdk动态代理 必须 实现接口：HandlerInterceptor

为什么拦截器一定实现一个接口： *HandlerInterceptor* ，是因为jdk动态代理必须要有一个接口，才能创建代理对象

定义拦截器

LogInterceptor实现HandlerInterceptor覆盖三个方法。然后把需要处理的日志在handler进行处理。

- 注册拦截器
- 配置拦截去路由

拦截器存在问题

- 和容器(servlet)有关的业务(request,response,freemarker)
- 粒度太粗了。因为拦截器是根据路由（RequestMapping）来决定，你只能拦截controller层，不能拦截和处理service层,dao层也就是controller以外的都没处理。

```

1 @PostMapping("/save/userorder")
2 public void saveUserOrder(){// --- 3s ~5s
3     // 保存订单
4     orderService.saveOrder(); // 需要记录日志 限流 100ms
5     // 用户积分增加
6     userService.updateUserjifen();//这个不需要 不做 800ms
7 }

```

- 逻辑处理不方便。太笨重了。
 - 如果出现异常呢（这个问题呢？）
 - 在方法执行中，
 - 在方法执行后
 - 执行结果之后
- 如果又多个处理，你必须定义多个拦截器，其实粒度太大了。配置和维护都灾难。

```

1 addRegisterInterceptor(new LogInterceptor())
2 .addPatterns("/user/**", "/order/**");
3
4 addRegisterInterceptor(new PermissionInterceptor())
5 .addPatterns("/user/save", "/user/update", "/order/makeorder");
6

```

jdk动态代理、cglib代码

使用aop的底层机制比如：jdk动态代理去实现或者cglib代理去实现。是没问题。但是面临如下问题：

 面临的问题：

- 精细的控制必须自己去编写，
 - 比如：条件判断，哪些方法要进，哪些方法不进。必须自己控制
 - 如果多个代理 代理对象的，代理类创建也必须自己定义（切面，连接点，切点，织入，通知增强，全部要自己去封装）
 - 多代理类怎么执行顺序是如何保证。你也需要自己控制
- 你必须对springioc的底层机制，和生命周期要非常的熟悉。因为你一定动态代理无论你是用jdk动态代理实现还是cglib代理实现，它们目标都是要把目标对象转化成：代理对象。
- 因为只有代理对象执行方法，才会进行到代理类（切面）去做逻辑增强处理。（我们处理）

07、AOP的重要性

- aop底层是jdk动态代理和cglib代理结合实现。
- 它可以解决上面所有的问题。
- aop它会自动把springioc中的增强对象全部自动转换成代理对象。

08、Aop的底层的实现机制

- jdk动态实现
- cglib代理实现

不论是用那种实现：它的目标都是一致的把springioc的对象转化代理对象。

这是一个正常springioc的对象

```
> this = {SpringbootAopApplicationTests@5989}
▼ userService = {UserService@5990}
> logService = {LogService@5992}
```

Variables

```
> this = {SpringbootAopApplicationTests@5989}
▼ orderService = {OrderService@5990}
> logService = {LogService@5992}
```

🍷 09、Springioc对象转化代理对象？

为什么用做日志处理，你日志拦截处理和核心点是什么？

- 查看当前类的执行时长
- 根据时长可以判断当前是那个方法耗时，可以根据这个耗时的方法针对性的排查和优化。

🍷 01、定义日志注解

```
1 package com.kuangstudy.aop;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 /**
9  * @author 飞哥
10  * @Title: 学相伴出品
11  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
12  * @date 2021/12/21 23:21
13  */
```



```

14 @Retention(RetentionPolicy.RUNTIME)
15 @Target({ElementType.METHOD})
16 public @interface KsdLog {
17
18     String value() default "";
19 }
20

```

为什么要定义一个注解：

- 它就相当于一个小红旗，那个方法需要被AOP拦截就写上注解，红旗就出现在方法上。
- 因为springaop的切点表达式做了注解的条件处理，可以控制加了注解的方法才会进行到通知（增强逻辑的位置）
- 不用注解可以吗？当然可以，但是不灵活，看场景
 - execution 表达式（事务控制）
 - within
 - target（针对性的）
 - args（针对的处理）
 - this
 - annatation(注解)--细粒度

02、定义切面

- @Component
 - 让springioc容器管理这个切面，变成一家人。
- @Aspect
 - 标记为切面
- 连接点
 - 切点表达式
- 通知
 - 前置Advice --- @Before(切点或者切点表达式)
 - 后置Advice --- @After(切点或者切点表达式)
 - 异常Advice -- @AfterThrowing(切点或者切点表达式)

- 环绕Advice ---- @Around(切点或者切点表达式)
- 后置返回Advice ----@AfterRetuning(切点或者切点表达式)

```
1 package com.kuangstudy.aop;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.aspectj.lang.ProceedingJoinPoint;
5 import org.aspectj.lang.Signature;
6 import org.aspectj.lang.annotation.*;
7 import org.aspectj.lang.reflect.MethodSignature;
8 import org.springframework.stereotype.Component;
9
10 /**
11  * @author 飞哥
12  * @Title: 学相伴出品
13  * @Description: 飞哥B站地址:
14  * https://space.bilibili.com/490711252
15  * 记得关注和三连哦!
16  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
17  * @date 2021/12/21 23:31
18  */
19 @Component
20 @Aspect
21 @Slf4j
22 public class LogAspect {
23     // 1: 定义切入点, 切入点标注有注解@KsdLog的所有函数, 通过
24     // @Pointcut 判断才可以进入到具体增强的通知
25     @Pointcut("@annotation(com.kuangstudy.aop.KsdLog)")
26     public void logpointcut() {
27     }
28
29     @Around("logpointcut()")
30     public void beforeAdvice(ProceedingJoinPoint
31     proceedingJoinPoint) throws Throwable {
32         try {
33             // 1: 方法执行的开始时间
```

```

33         long starttime = System.currentTimeMillis();
34         // 2:执行真实方法
35         MethodSignature signature = (MethodSignature)
proceedingJoinPoint.getSignature();
36
37         System.out.println(signature.getMethod().getName());
38         System.out.println(signature.getReturnType());
39
40         System.out.println(signature.getParameterNames());
41         System.out.println(signature.getParameterTypes());
42         Object methodReturnValue =
proceedingJoinPoint.proceed();
43         log.info("当前执行方法的返回值是: {}",
methodReturnValue);
44         // 3:方法执行的结束时间
45         long endtime = System.currentTimeMillis();
46         // 4: 方法的总耗时
47         long total = endtime - starttime;
48         log.info("当前方法:{}", 执行的时间是: {} ms",
signature.getMethod().getName(), total);
49
50     } catch (Throwable ex) {
51         log.info("执行方法出错.....,{}", ex);
52         throw ex;
53     }
54 }
55

```

在需要拦截的方法上增加@KsdLog注解

```

1 package com.kuangstudy.first;
2
3 import com.kuangstudy.aop.KsdLog;
4 import lombok.extern.slf4j.Slf4j;

```

```
5  import
   org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  /**
9   * @author 飞哥
10  * @Title: 学相伴出品
11  * @Description: 飞哥B站地址:
   https://space.bilibili.com/490711252
12  * 记得关注和三连哦!
13  * @Description: 我们有一个学习网站: https://www.kuangstudy.com
14  * @date 2021/12/21 22:05
15  */
16  @Slf4j
17  @Service
18  public class OrderService {
19
20      @Autowired
21      private LogService logService;
22
23      /**
24       * 订单注册
25       *
26       * @param order
27       */
28      @KsdLog
29      public void saveOrder(Order order) {
30          log.info("订单注册....");
31
32          // 记录日志
33          Logs logs = new Logs();
34
35          logs.setClassname("com.kuangstudy.service.OrderService");
36          logs.setMethod("saveOrder");
37          logs.setId(1);
38          logs.setParams(order.toString());
39          logs.setTime("100ms");
39          logService.saveLog(logs);
```

```
40     }
41
42     /**
43      * 修改订单
44      *
45      * @param order
46      */
47     @KsdLog
48     public void updateOrder(Order order) {
49         log.info("修改订单....");
50
51         // 记录日志
52         Logs logs = new Logs();
53
54         logs.setClassname("com.kuangstudy.service.OrderService");
55         logs.setMethod("updateOrder");
56         logs.setId(1);
57         logs.setParams(order.toString());
58         logs.setTime("100ms");
59         logService.saveLog(logs);
60     }
61
62     /**
63      * 删除订单
64      *
65      * @param orderId
66      */
67     @KsdLog
68     public void delOrder(Integer orderId) {
69         log.info("删除订单....{}", orderId);
70
71         // 记录日志
72         Logs logs = new Logs();
73
74         logs.setClassname("com.kuangstudy.service.OrderService");
75         logs.setMethod("delOrder");
76         logs.setId(1);
```

```

76         logs.setParams(orderId.toString());
77         logs.setTime("100ms");
78         logService.saveLog(logs);
79     }
80
81 }
82

```

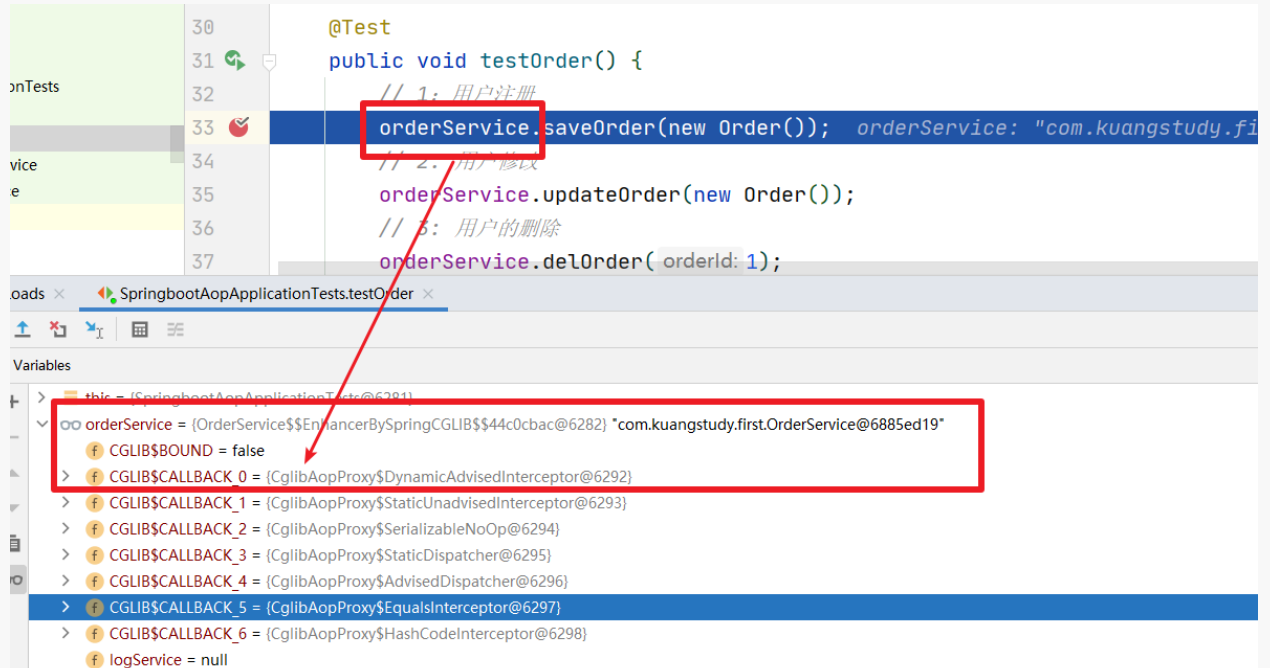
运行测试代码如下：

这个时候我们很清晰的看orderService变成了代理对象

```

variables
> this = {SpringbootAopApplicationTests@5989}
▼ orderService = {OrderService@5990}
> logService = {LogService@5992}

```



```

30 @Test
31 public void testOrder() {
32     // 1. 用户注册
33     orderService.saveOrder(new Order()); orderService: "com.kuangstudy.fi
34     // 2. 用户修改
35     orderService.updateOrder(new Order());
36     // 3. 用户的删除
37     orderService.delOrder( orderId: 1);

```

```

Variables
> this = {SpringbootAopApplicationTests@6281}
▼ orderService = {OrderService$$EnhancerBySpringCGLIB$$44c0cbac@6282} *com.kuangstudy.first.OrderService@6885ed19"
  CGLIB$BOUND = false
  CGLIB$CALLBACK_0 = {CglibAopProxy$DynamicAdvisedInterceptor@6292}
  CGLIB$CALLBACK_1 = {CglibAopProxy$StaticUnadvisedInterceptor@6293}
  CGLIB$CALLBACK_2 = {CglibAopProxy$SerializableNoOp@6294}
  CGLIB$CALLBACK_3 = {CglibAopProxy$StaticDispatcher@6295}
  CGLIB$CALLBACK_4 = {CglibAopProxy$AdvisedDispatcher@6296}
  CGLIB$CALLBACK_5 = {CglibAopProxy$EqualsInterceptor@6297}
  CGLIB$CALLBACK_6 = {CglibAopProxy$HashCodeInterceptor@6298}
  logService = null

```

10、结论

- 在spring底层框架中，它是ioc做核心基础，ioc做的管理项目所有的bean对象。如果这些管理的bean中的任何一个方法只要和aop的切面有管理。就会把这些bean全部转换成代理对象。
- 因为只有代理对象执行方法，才可以进入到代理类（切面）中去做逻辑增强处理
- 而哪些如果没用加满足条件的方法，全部会自然抛弃和正常执行，而满足条件的就进入对应通知中进行逻辑增强。
- springboot默认情况下代码类型是：cglib代理，而spring的框架默认代理：jdk动态代理。
 - 因为jdk动态代理必须要接口。
 - 但是在有些情况下，开发不需要接口，依然要能够支持代理，现如用jdk动态代理就无法上西安
 - 为解决这种众口难调的情况，有些喜欢面向类变成，有人喜欢面向接口编程spring都做了兼容和支持。