|  |
| --- |
| Spring |
| AOP SpringAOP-24 myTestAop  Aop mitHilfe von context.xml  public class Start {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");    MyService myService = (MyService) context.getBean("myService");  try {  myService.divide(0, 0);  } catch (Exception e) {  System.*out*.println("ich main method habe gerade eine FEHLER gefangen");  }  myService.testAspekt2();  myService.testAspekt();  myService.retOb();  }  }  --------------------------------------------------------------------------------------------------------------------  @Component  public class MyLoggerV2 {    public void init()  {  System.*out*.println("Ich \"MyLoggerV2\" führe Methode init() aus ");  }    public void printo(Object objx)  {  System.*out*.println("Ich \"MyLoggerV2\" führe Methode printo() aus und drucke dem zurückgegebenen Object aus -"+objx);    }    public void exc(Exception ex)  {  System.*out*.println("Ich \"MyLoggerV2\" habe gerade eine Fehler gefangen ");    }    public void aroundmy(ProceedingJoinPoint joinpoiunt){    System.*out*.println("Ich \"MyLoggerV2\" führe mich aus befor der Mothode "+ joinpoiunt.getSignature().toShortString());    try {  joinpoiunt.proceed();  } catch (Throwable e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  System.*out*.println("Ich \"MyLoggerV2\" führe mich aus after der Mothode "+ joinpoiunt.getSignature().toShortString());        }  }  ----------------------------------------------------------------  @Component  public class MyService {  public void testAspekt() {  System.*out*.println("ich MyService fürhe Methode "+methodenName());  }  private String methodenName() {  StackTraceElement stackTraceElement = Thread.*currentThread*().getStackTrace()[2];  String klassenName = stackTraceElement.getClassName();  String methodenName = stackTraceElement.getMethodName();    return methodenName;  }  public void testAspekt2() {  System.*out*.println("ich MyService fürhe Methode "+methodenName());  }  public Object retOb() {  System.*out*.println("ich MyService fürhe Methode "+methodenName());  return this;  }  public void divide(double v, double b) throws Exception {  System.*out*.println("ich MyService fürhe Methode "+methodenName()+" aus die ein Fehler generiert");  throw new NullPointerException("Herrrrr tebe");  }  }  -------------------------------------------------------------------------------------------  <context:component-scan base-package=*"ru.javabegin.training.\*"* />  <aop:config>    <aop:aspect id=*"myAspect"* ref=*"myLoggerV2"*>  <aop:pointcut id=*"myValue"*  expression=*"execution(\* ru.javabegin.training.spring.aop.objects.MyService.\*(..))"* />  <aop:before pointcut-ref=*"myValue"* method=*"init"* />  <aop:after-returning pointcut-ref=*"myValue"*  returning=*"objx"* method=*"printo"* />  <aop:after-throwing pointcut-ref=*"myValue"*  throwing=*"ex"* method=*"exc"* />    </aop:aspect>  <aop:aspect id=*"myAspect2"* ref=*"myLoggerV2"*>  <aop:pointcut id=*"myValue2"*  expression=*"execution(\* ru.javabegin.training.spring.aop.objects.MyService.testAspekt2(..))"* />  <aop:around method=*"aroundmy"* pointcut-ref=*"myValue2"* />  </aop:aspect>  </aop:config>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>4.0.3.RELEASE</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.7.4</version>  </dependency>  AOP SpringAOP-24 myTestAopAspect  Aop mitHilfe von Annotation  public class Start {  public static void main(String[] args) {      ApplicationContext appContext= new ClassPathXmlApplicationContext("context.xml");  TestService myservice=(TestService) appContext.getBean("testService");    myservice.getInfo();  myservice.getObject();  }  }  --------------------------------------------------------------------------  @Component  @Aspect  public class MyAopService {    @Pointcut("execution(\* myTestAop.TestService.\*())")  public void AllMethod()  {    }    @Around("AllMethod()")  public Object myService(ProceedingJoinPoint joinpoiunt){    System.*out*.println("ich bin service");  Object obj=null;        try {  obj=joinpoiunt.proceed();  } catch (Throwable e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    System.*out*.println(obj.getClass());  return obj;  }      @AfterReturning(pointcut="AllMethod()", returning="obj" )  public void tesAfterRet(Object obj){    System.*out*.println(obj.getClass().getSimpleName());  }  }  --------------------------------------------------------------------------  @Component  public class TestService {  public int getInfo()  {  System.*out*.println("ich bin mei getInfo");  return 2;  }    //  public int getObject(){    System.*out*.println("ich bin mei getObject");    return 3;  }  }  AOP own interfaces myTestAopAspectOwnInterface  @Retention(RetentionPolicy.*RUNTIME*)  @Target(ElementType.*METHOD*)  public @interface IMyType {  }  --------------------------------------  @After("AllMethod() && @annotation(IMyType)")  public void testAnnotation()  {  System.*out*.println("das klappt");  }  --------------------------------------  @IMyType  public int testAnnotation(){    System.*out*.println("TestAnnotation");    return 11;  }  Pointcut für alle methoden die ein Interface implementieren myTestAopAspectOwnInterface  public interface myInterface {    String getText();  }  -----------------------------------  @Override  public String getText() {  // TODO Auto-generated method stub  return "hallo";  }  ..............................  @Pointcut("execution(\* myTestAop.myInterface.\*(..))")  public void AllMethodWithInterface()  {    }  .......................................  @After("AllMethodWithInterface()")  public void methodInterface()  {  System.*out*.println("Interface klappt");  }  -----------------------------------------------  <aop:aspectj-autoproxy proxy-target-class=*"true"*></aop:aspectj-autoproxy> |
| Pointcut für alle methoden die ein Integer zurückgeben  @Pointcut("execution(java.lang.Integer \*(..))")  public void AllMethodReturnInteger()  {    } |
| Pointcut to find Method witch argument int  @Pointcut("execution(java.lang.Integer \*(int))")  public void AllMethodReturnInteger()  {      }    @Before ("AllMethodReturnInteger() && args(myInt)")  public void returningInteger(int myInt){    System.*out*.println("Integer klappt " + myInt);  } |