

课程目标

- 1、高仿真手写 Spring AOP 部分，充分实践设计模式。
- 2、用 30 个类搭建基本框架，满足核心功能。

内容定位

在完全掌握 Spring 系统结构、实现原理，在理解设计模式的基础上，自己动手写一个高仿真版本的 Spring 框架，以达到透彻理解 Spring 的目的，感受作者创作意图。

基础配置

在 application.properties 中增加如下自定义配置：

```
#多切面配置可以在 key 前面加前缀
#例如 aspect.logAspect.

#切面表达式#
pointCut=public .* com.gupaoedu.vip.spring.demo.service.*Service..*(.*)
#切面类#
aspectClass=com.gupaoedu.vip.spring.demo.aspect.LogAspect
#切面前置通知#
aspectBefore=before
#切面后置通知#
aspectAfter=after
#切面异常通知#
aspectAfterThrow=afterThrowing
#切面异常类型#
aspectAfterThrowingName=java.lang.Exception
```

下面是 Spring AOP 的原生配置，为了方便操作，用 properties 文件来代替 xml，以简化操作：

```
<bean id="xmlAspect" class="com.gupaoedu.aop.aspect.XmlAspect"></bean>

<!-- AOP 配置 -->
<aop:config>
```

```

<!-- 声明一个切面,并注入切面 Bean,相当于@Aspect -->
<aop:aspect ref="xmlAspect">
    <!-- 配置一个切入点,相当于@Pointcut -->
    <aop:pointcut expression="execution(* com.gupaoedu.aop.service..*(..))"
id="simplePointcut"/>
    <!-- 配置通知,相当于@Before、@After、@AfterReturn、@Around、@AfterThrowing -->
    <aop:before pointcut-ref="simplePointcut" method="before"/>
    <aop:after pointcut-ref="simplePointcut" method="after"/>
    <aop:after-returning pointcut-ref="simplePointcut" method="afterReturn"/>
    <aop:after-throwing pointcut-ref="simplePointcut" method="afterThrow" throwing="ex"/>
</aop:aspect>

</aop:config>

```

完成 AOP 顶层设计

GPAopProxy

```

package com.gupaoedu.vip.spring.framework.aop;

/**
 * Created by Tom.
 */
//默认就用 JDK 动态代理
public interface GPAopProxy {
    Object getProxy();
    Object getProxy(ClassLoader classLoader);
}

```

GPCglibAopProxy

```

package com.gupaoedu.vip.spring.framework.aop;

import com.gupaoedu.vip.spring.framework.aop.support.GPAdvisedSupport;

/**
 * Created by Tom.
 */
public class GPCglibAopProxy implements GPAopProxy {
    private GPAdvisedSupport config;

    public GPCglibAopProxy(GPAdvisedSupport config){
        this.config = config;
    }
}

```

```

@Override
public Object getProxy() {
    return null;
}

@Override
public Object getProxy(ClassLoader classLoader) {
    return null;
}
}

```

GPJdkDynamicAopProxy

```

package com.gupaoedu.vip.spring.formework.aop;

import com.gupaoedu.vip.spring.formework.aop.intercept.GPMethodInvocation;
import com.gupaoedu.vip.spring.formework.aop.support.GPAdvisedSupport;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.util.List;

/**
 * Created by Tom.
 */
public class GPJdkDynamicAopProxy implements GPAopProxy, InvocationHandler {
    private GPAdvisedSupport config;

    public GPJdkDynamicAopProxy(GPAdvisedSupport config){
        this.config = config;
    }

    //把原生的对象传进来
    public Object getProxy(){
        return getProxy(this.config.getTargetClass().getClassLoader());
    }

    @Override
    public Object getProxy(ClassLoader classLoader) {
        return
        Proxy.newProxyInstance(classLoader, this.config.getTargetClass().getInterfaces(), this);
    }
}

```

```

@Override
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    List<Object> interceptorsAndDynamicMethodMatchers =
config.getInterceptorsAndDynamicInterceptionAdvice(method,this.config.getTargetClass());
    GPMethodInvocation invocation = new
GPMethodInvocation(proxy,this.config.getTarget(),method,args,this.config.getTargetClass(),interc
eptorsAndDynamicMethodMatchers);
    return invocation.proceed();
}
}

```

GPAvisedSupport

```

package com.gupaoedu.vip.spring.formework.aop.support;

import com.gupaoedu.vip.spring.formework.aop.GPAopConfig;
import com.gupaoedu.vip.spring.formework.aop.aspect.GPAfterReturningAdvice;
import com.gupaoedu.vip.spring.formework.aop.aspect.GPAfterThrowingAdvice;
import com.gupaoedu.vip.spring.formework.aop.aspect.GPMethodBeforeAdvice;

import java.lang.reflect.Method;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Created by Tom.
 */
public class GPAvisedSupport {
    private Class targetClass;
    private Object target;
    private Pattern pointCutClassPattern;

    private transient Map<Method, List<Object>> methodCache;

    private GPAopConfig config;

    public GPAvisedSupport(GPAopConfig config){
        this.config = config;
    }

    public Class getTargetClass() {
        return targetClass;
    }

```

```

}

public void setTargetClass(Class targetClass) {
    this.targetClass = targetClass;
    parse();
}

public Object getTarget() {
    return target;
}

public void setTarget(Object target) {
    this.target = target;
}

public List<Object> getInterceptorsAndDynamicInterceptionAdvice(Method method, Class<?>
targetClass) throws Exception {
    List<Object> cached = methodCache.get(method);

    // 缓存未命中，则进行下一步处理
    if (cached == null) {
        Method m = targetClass.getMethod(method.getName(), method.getParameterTypes());
        cached = methodCache.get(m);
        // 存入缓存
        this.methodCache.put(m, cached);
    }
    return cached;
}

public boolean pointCutMatch(){
    return pointCutClassPattern.matcher(this.targetClass.toString()).matches();
}

private void parse(){
    //pointCut 表达式
    String pointCut = config.getPointCut()
        .replaceAll("\\.", "\\\\.")
        .replaceAll("\\\\.*", ".*")
        .replaceAll("\\(", "\\(")
        .replaceAll("\\)", "\\)");

    String pointCutForClass = pointCut.substring(0, pointCut.lastIndexOf("\\(") - 4);
    pointCutClassPattern = Pattern.compile("class " +
pointCutForClass.substring(pointCutForClass.lastIndexOf(" ") + 1));

```

```

methodCache = new HashMap<Method, List<Object>>();
Pattern pattern = Pattern.compile(pointCut);

try {
    Class aspectClass = Class.forName(config.getAspectClass());
    Map<String, Method> aspectMethods = new HashMap<String, Method>();
    for (Method m : aspectClass.getMethods()){
        aspectMethods.put(m.getName(), m);
    }

    //在这里得到的方法都是原生的方法
    for (Method m : targetClass.getMethods()){

        String methodString = m.toString();
        if(methodString.contains("throws")){
            methodString =
methodString.substring(0, methodString.lastIndexOf("throws")).trim();
        }
        Matcher matcher = pattern.matcher(methodString);
        if(matcher.matches()){
            //能满足切面规则的类，添加的 AOP 配置中
            List<Object> advices = new LinkedList<Object>();
            //前置通知
            if(!(null == config.getAspectBefore() ||
"".equals(config.getAspectBefore().trim()))){
                advices.add(new
GPMMethodBeforeAdvice(aspectMethods.get(config.getAspectBefore()), aspectClass.newInstance()));
            }
            //后置通知
            if(!(null == config.getAspectAfter() ||
"".equals(config.getAspectAfter().trim()))){
                advices.add(new
GPAAfterReturningAdvice(aspectMethods.get(config.getAspectAfter()), aspectClass.newInstance()));
            }
            //异常通知
            if(!(null == config.getAspectAfterThrow() ||
"".equals(config.getAspectAfterThrow().trim()))){
                GPAfterThrowingAdvice afterThrowingAdvice = new
GPAfterThrowingAdvice(aspectMethods.get(config.getAspectAfterThrow()),
aspectClass.newInstance());
                afterThrowingAdvice.setThrowingName(config.getAspectAfterThrowingName());
                advices.add(afterThrowingAdvice);
            }
        }
    }
}

```

```

        methodCache.put(m,advices);
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

GPMethodInvocation

```

package com.gupaoedu.vip.spring.framework.aop.intercept;

import com.gupaoedu.vip.spring.framework.aop.aspect.GPJoinPoint;

import java.lang.reflect.Method;
import java.util.List;

/**
 * Created by Tom.
 */
public class GPMethodInvocation implements GPJoinPoint {

    private Object proxy;
    private Method method;
    private Object target;
    private Class<?> targetClass;
    private Object[] arguments;
    private List<Object> interceptorsAndDynamicMethodMatchers;

    private int currentInterceptorIndex = -1;

    public GPMethodInvocation(Object proxy, Object target, Method method, Object[] arguments,
                             Class<?> targetClass, List<Object>
interceptorsAndDynamicMethodMatchers) {
        this.proxy = proxy;
        this.target = target;
        this.targetClass = targetClass;
        this.method = method;
        this.arguments = arguments;
        this.interceptorsAndDynamicMethodMatchers = interceptorsAndDynamicMethodMatchers;
    }

    public Object proceed() throws Throwable {

```

```

        if (this.currentInterceptorIndex == this.interceptorsAndDynamicMethodMatchers.size() - 1)
        {
            return this.method.invoke(this.target,this.arguments);
        }
        Object interceptorOrInterceptionAdvice =
            this.interceptorsAndDynamicMethodMatchers.get(++this.currentInterceptorIndex);
        if (interceptorOrInterceptionAdvice instanceof GPMMethodInterceptor) {
            GPMMethodInterceptor mi = (GPMMethodInterceptor) interceptorOrInterceptionAdvice;
            return mi.invoke(this);
        } else {
            return proceed();
        }

    }

    @Override
    public Method getMethod() {
        return this.method;
    }

    @Override
    public Object[] getArguments() {
        return this.arguments;
    }

    @Override
    public Object getThis() {
        return this.target;
    }
}

```

GPMMethodInterceptor

```

package com.gupaoedu.vip.spring.framework.aop.intercept;

/**
 * Created by Tom.
 */
public interface GPMMethodInterceptor{
    Object invoke(GPMMethodInvocation mi) throws Throwable;
}

```

GPAvice

```

/**
 * Created by Tom.

```



```

*/
public interface GPAdvice {

}

```

GPJoinPoint

```

package com.gupaoedu.vip.spring.framework.aop.aspect;

import java.lang.reflect.Method;

/**
 * Created by Tom.
 */
public interface GPJoinPoint {

    Method getMethod();

    Object[] getArguments();

    Object getThis();

}

```

GPAopConfig

```

package com.gupaoedu.vip.spring.framework.aop;
import lombok.Data;

/**
 * Created by Tom.
 */
@Data
public class GPAopConfig {
    private String pointCut;
    private String aspectBefore;
    private String aspectAfter;
    private String aspectClass;
    private String aspectAfterThrow;
    private String aspectAfterThrowingName;
}

```

设计 AOP 基础实现

GPAbstractAspectJAdvice

```

package com.gupaoedu.vip.spring.formework.aop.aspect;

import java.lang.reflect.Method;

/**
 * Created by Tom.
 */
public abstract class GPAbstractAspectJAdvice implements GAdvice {

    private Method aspectMethod;
    private Object aspectTarget;

    public GPAbstractAspectJAdvice(
        Method aspectMethod, Object aspectTarget) {
        this.aspectMethod = aspectMethod;
        this.aspectTarget = aspectTarget;
    }

    protected Object invokeAdviceMethod(GPJoinPoint joinPoint, Object returnValue, Throwable ex)
        throws Throwable {
        Class<?> [] paramTypes = this.aspectMethod.getParameterTypes();
        if (null == paramTypes || paramTypes.length == 0) {
            return this.aspectMethod.invoke(aspectTarget);
        } else {
            Object[] args = new Object[paramTypes.length];
            for (int i = 0; i < paramTypes.length; i++) {
                if (paramTypes[i] == GPJoinPoint.class) {
                    args[i] = joinPoint;
                } else if (paramTypes[i] == Throwable.class) {
                    args[i] = ex;
                } else if (paramTypes[i] == Object.class) {
                    args[i] = returnValue;
                }
            }
            return this.aspectMethod.invoke(aspectTarget, args);
        }
    }
}

```

GPMMethodBeforeAdvice

```

package com.gupaoedu.vip.spring.formework.aop.aspect;

import com.gupaoedu.vip.spring.formework.aop.intercept.GPMMethodInterceptor;
import com.gupaoedu.vip.spring.formework.aop.intercept.GPMMethodInvocation;

```

```

import java.lang.reflect.Method;

/**
 * Created by Tom.
 */
public class GPMethodBeforeAdvice extends GPAbstractAspectJAdvice implements
GPAAdvice, GPMethodInterceptor {

    private GPJoinPoint joinPoint;

    public GPMethodBeforeAdvice(Method aspectMethod, Object target) {
        super(aspectMethod, target);
    }

    public void before(Method method, Object[] args, Object target) throws Throwable {
        invokeAdviceMethod(this.joinPoint, null, null);
    }

    public Object invoke(GPMethodInvocation mi) throws Throwable {
        this.joinPoint = mi;
        this.before(mi.getMethod(), mi.getArguments(), mi.getThis());
        return mi.proceed();
    }
}

```

GPAfterReturningAdvice

```

package com.gupaoedu.vip.spring.formework.aop.aspect;

import com.gupaoedu.vip.spring.formework.aop.intercept.GPMethodInterceptor;
import com.gupaoedu.vip.spring.formework.aop.intercept.GPMethodInvocation;

import java.lang.reflect.Method;

/**
 * Created by Tom.
 */
public class GPAfterReturningAdvice extends GPAbstractAspectJAdvice implements
GPAAdvice, GPMethodInterceptor {

    private GPJoinPoint joinPoint;
    public GPAfterReturningAdvice(Method aspectMethod, Object target) {
        super(aspectMethod, target);
    }
}

```

```

@Override
public Object invoke(GPMethodInvocation mi) throws Throwable {
    Object retVal = mi.proceed();
    this.joinPoint = mi;
    this.afterReturning(retVal, mi.getMethod(), mi.getArguments(), mi.getThis());
    return retVal;
}

public void afterReturning(Object returnValue, Method method, Object[] args, Object target) throws
Throwable{
    invokeAdviceMethod(joinPoint, returnValue, null);
}
}

```

GPAfterThrowingAdvice

```

package com.gupaoedu.vip.spring.formework.aop.aspect;

import com.gupaoedu.vip.spring.formework.aop.intercept.GPMethodInterceptor;
import com.gupaoedu.vip.spring.formework.aop.intercept.GPMethodInvocation;

import java.lang.reflect.Method;

/**
 * Created by Tom.
 */
public class GPAfterThrowingAdvice extends GPAbstractAspectJAdvice implements
GPAdvice, GPMethodInterceptor {

    private String throwingName;
    private GPMethodInvocation mi;

    public GPAfterThrowingAdvice(Method aspectMethod, Object target) {
        super(aspectMethod, target);
    }

    public void setThrowingName(String name) {
        this.throwingName = name;
    }

    @Override
    public Object invoke(GPMethodInvocation mi) throws Throwable {
        try {

```

```

        return mi.proceed();
    }catch (Throwable ex) {
        invokeAdviceMethod(mi,null,ex.getCause());
        throw ex;
    }
}
}
}

```

接入 getBean()方法

找到 GPApplicationContext 的 getBean()方法，我们知道 getBean()中负责 Bean 初始化的方法其实就是 instantiateBean()，我们在初始化时就可以确定是否返回原生 Bean 还是 Proxy Bean。代码实现如下：

```

//传一个 BeanDefinition，就返回一个实例 Bean
private Object instantiateBean(GPBeanDefinition beanDefinition){
    Object instance = null;
    String className = beanDefinition.getBeanClassName();
    try{

        //因为根据 Class 才能确定一个类是否有实例
        if(this.singletonBeanCacheMap.containsKey(className)){
            instance = this.singletonBeanCacheMap.get(className);
        }else{
            Class<?> clazz = Class.forName(className);
            instance = clazz.newInstance();

            GPAdvisedSupport config = instantiationAopConfig(beanDefinition);
            config.setTargetClass(clazz);
            config.setTarget(instance);

            if(config.pointCutMatch()) {
                instance = createProxy(config).getProxy();
            }

            this.singletonBeanCacheMap.put(beanDefinition.getFactoryBeanName(),instance);
        }

        return instance;
    }catch (Exception e){
        e.printStackTrace();
    }
}

```

```

        return null;
    }

    private GPointCutSupport instantiationAopConfig(GPBeanDefinition beanDefinition) throws Exception{

        GPointCutSupport config = new GPointCutSupport();
        config.setPointCut(reader.getConfig().getProperty("pointCut"));
        config.setAspectClass(reader.getConfig().getProperty("aspectClass"));
        config.setAspectBefore(reader.getConfig().getProperty("aspectBefore"));
        config.setAspectAfter(reader.getConfig().getProperty("aspectAfter"));
        config.setAspectAfterThrow(reader.getConfig().getProperty("aspectAfterThrow"));

        config.setAspectAfterThrowingName(reader.getConfig().getProperty("aspectAfterThrowingName"));

        return new GPointCutSupport(config);
    }

    private GPointCutProxy createProxy(GPointCutSupport config) {
        Class targetClass = config.getTargetClass();
        if (targetClass.getInterfaces().length > 0) {
            return new GJdkDynamicAopProxy(config);
        }
        return new GCGlibAopProxy(config);
    }
}

```

织入业务代码

LogAspect

```

package com.gupaoedu.vip.spring.demo.aspect;

import com.gupaoedu.vip.spring.framework.aop.aspect.GPJoinPoint;
import lombok.extern.slf4j.Slf4j;

import java.util.Arrays;

/**
 * Created by Tom.
 */
@Slf4j
public class LogAspect {

    //在调用一个方法之前，执行 before 方法
    public void before(GPJoinPoint joinPoint){

```

```

//这个方法中的逻辑，是由我们自己写的
Log.info("Invoker Before Method!!!" +
        "\nTargetObject:" + joinPoint.getTarget() +
        "\nArgs:" + Arrays.toString(joinPoint.getArguments()));
}

//在调用一个方法之后，执行 after 方法
public void after(GPJoinPoint joinPoint){
    Log.info("Invoker After Method!!!" +
            "\nTargetObject:" + joinPoint.getTarget() +
            "\nArgs:" + Arrays.toString(joinPoint.getArguments()));
}

public void afterThrowing(GPJoinPoint joinPoint, Throwable ex){
    Log.info("出现异常" +
            "\nTargetObject:" + joinPoint.getTarget() +
            "\nArgs:" + Arrays.toString(joinPoint.getArguments()) +
            "\nThrows:" + ex.getMessage());
}
}

```

IModifyService

```

package com.gupaoedu.vip.spring.demo.service;

/**
 * 增删改业务
 * @author Tom
 *
 */
public interface IModifyService {

    /**
     * 增加
     */
    String add(String name, String addr) throws Exception;

    /**
     * 修改
     */
    String edit(Integer id, String name);

    /**
     * 删除
     */
}

```

```
String remove(Integer id);

}
```

ModifyService

```
package com.gupaoedu.vip.spring.demo.service.impl;

import com.gupaoedu.vip.spring.demo.service.IModifyService;
import com.gupaoedu.vip.spring.formework.annotation.GPService;

/**
 * 增删改业务
 * @author Tom
 *
 */
@GPService
public class ModifyService implements IModifyService {

    /**
     * 增加
     */
    public String add(String name,String addr) throws Exception {
        throw new Exception("故意抛出异常，测试切面通知是否生效");
        // return "modifyService add,name=" + name + ",addr=" + addr;
    }

    /**
     * 修改
     */
    public String edit(Integer id,String name) {
        return "modifyService edit,id=" + id + ",name=" + name;
    }

    /**
     * 删除
     */
    public String remove(Integer id) {
        return "modifyService id=" + id;
    }
}
```

运行效果演示

输入： /web/add.json?name=Tom&addr=HunanChangsha



控制台输出：

```

2019-04-14 16:39:53.964 [925687314@qtp-974606690-3] INFO com.gupaoedu.vip.spring.demo.aspect.LogAspect - Invoker Before Method!!!
TargetObject:com.gupaoedu.vip.spring.demo.service.impl.ModifyService@10d2a754
Args:[Tom, HunanChangsha]
2019-04-14 16:39:53.964 [925687314@qtp-974606690-3] INFO com.gupaoedu.vip.spring.demo.aspect.LogAspect - 出现异常
TargetObject:com.gupaoedu.vip.spring.demo.service.impl.ModifyService@10d2a754
Args:[Tom, HunanChangsha]
Throws:故意抛出异常，测试切面通知是否生效

```

从控制台输出可以看到切面已经生效了，触发了 GPMethodBeforeAdvice 和 GPAfterThrowingAdvice，未触发 GPAfterReturningAdvice。

下面再做一个测试，输入： /web/add.json?name=Tom&addr=HunanChangsha



```
{name:"Tom", time:"2019-04-14 16:45:08"}
```

控制台输出：

```

2019-04-14 16:45:08.525 [1934313971@qtp-974606690-5] INFO com.gupaoedu.vip.spring.demo.aspect.LogAspect - Invoker Before Method!!!
TargetObject:com.gupaoedu.vip.spring.demo.service.impl.QueryService@6cf33f57
Args:[Tom]
2019-04-14 16:45:08.526 [1934313971@qtp-974606690-5] INFO c.g.vip.spring.demo.service.impl.QueryService - 这是在业务方法中打印的: {name:"Tom", time:"2019-04-14 16:45:08"}
2019-04-14 16:45:08.527 [1934313971@qtp-974606690-5] INFO com.gupaoedu.vip.spring.demo.aspect.LogAspect - Invoker After Method!!!
TargetObject:com.gupaoedu.vip.spring.demo.service.impl.QueryService@6cf33f57
Args:[Tom]

```

至此 AOP 模块就大功告成。