45 Spring中多个AOP如何协调执行？

更新时间：2020-09-02 10:26:09

> 理想必须要人们去实现它，它不但需要决心和勇敢而且需要知识。——吴玉章

背景

通过前面的章节，我们知道在程序开发中 AOP 主要用来解决一些系统层面上的问题，Struts2 的拦截器设计就是基于 AOP 的思想，是个比较经典的例子。总结一下 AOP 通用的使用场景主要有以下几方面：

```
Authentication 权限
Caching 缓存
Context passing 内容传递
Error handling 错误处理
Lazy loading　懒加载
Debugging　　调试
logging, tracing, profiling and monitoring　记录跟踪　优化　校准
Performance optimization　性能优化
Persistence　　持久化
Resource pooling　资源池
Synchronization　同步
Transactions 事务
```

在实际的环境中，AOP 被广泛使用，故可能存在同一个切入点（joint point）有多个 aspect 的情况，此时如果不协调他们的执行顺序，执行结果就是不可控的，那么如何协调这个 aspect 按照你设定的顺序执行呢？

Aspect 排序示例

Spring AOP 不支持 Advice 层级的排序，只支持 Aspect 层级的排序。不同的 Aspect 通过实现 Ordered 接口，或使用 @Order 注解设置优先级。对于 getOrder 返回的值或 @Order 中设置的值，**值越小优先级越高**。

方式一：**aspect 实现 Ordered 接口**

目标类：

```
package com.davidwang456.test;
public class HelloService {
 public void sayHello(String world) {
  System.out.println("hello "+ world);
 }
}
```

切面类：

针对 sayHello 方法做前置处理的 Aspect1，其 order 为 5：

```
package com.davidwang456.test;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import org.springframework.core.Ordered;

@Aspect
public class LogAspect implements Ordered{

 @Before("execution(* sayHello(..))")
 public void beforeHello() {
  System.out.println("how are you !");
 }

 @Override
 public int getOrder() {
  return 5;
 }
}
```

重写了getOrder()方法。

针对sayHello方法做前置处理的Aspect1,其order为5：

```
package com.davidwang456.test;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import org.springframework.core.Ordered;

@Aspect
public class LogAspect2 implements Ordered{
 @Before("execution(* sayHello(..))")
 public void beforeHello() {
  System.out.println("how do you do !");
 }

 @Override
 public int getOrder() {
  return 10;
 }
}
```

测试类：

使用 AspectJProxyFactory 硬编码来生成代理类进行测试:

```
package com.davidwang456.test;

import org.springframework.aop.aspectj.annotation.AspectJProxyFactory;

public class AspectTest {
 public static void main(String[] args) {
  HelloService target=new HelloService();
  AspectJProxyFactory factory=new AspectJProxyFactory();
  factory.setTarget(target);
  factory.addAspect(LogAspect.class);
  factory.addAspect(LogAspect2.class);
  HelloService proxy=factory.getProxy();
  proxy.sayHello("world !");
  //ApplicationContext context= new ClassPathXmlApplicationContext("com/davidwang456/test/spring.xml");
  //HelloService hello=context.getBean(HelloService.class);
  //hello.sayHello("world !");
 }
}
```

运行结果：

```
how are you!

how do you do!

hello world !
```

也可以使用 xml 配置的方式，如注释的代码。

xml 配置文件如下：

```
<?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
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
 http://www.springframework.org/schema/aop
 http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

 <aop:aspectj-autoproxy />

 <!-- Target -->
 <bean id="hello" class="com.davidwang456.test.HelloService" />

 <!-- Aspect -->
 <bean id="logAspect" class="com.davidwang456.test.LogAspect" />

 <!-- Aspect -->
 <bean id="logAspect2" class="com.davidwang456.test.LogAspect2" />

</beans>
```

<aop:aspectj-autoproxy /> 开启 AspectJ 主动代理。执行结果如硬编码一致。

改变一下第一个切面 LogAspect 的顺序，order 变为 15：

```
package com.davidwang456.test;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import org.springframework.core.Ordered;

@Aspect
public class LogAspect implements Ordered{
 @Before("execution(* sayHello(..))")
 public void beforeHello() {
  System.out.println("how are you !");
 }

 @Override
 public int getOrder() {
  return 15;
 }
}
```

再次运行测试程序：

```
package com.davidwang456.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AspectTest {
 public static void main(String[] args) {
  //HelloService target=new HelloService();
  //AspectJProxyFactory factory=new AspectJProxyFactory();
  //factory.setTarget(target);
  //factory.addAspect(LogAspect.class);
  //factory.addAspect(LogAspect2.class);
  //HelloService proxy=factory.getProxy();
  //proxy.sayHello("world !");
  ApplicationContext context= new ClassPathXmlApplicationContext("com/davidwang456/test/spring.xml");
  HelloService hello=context.getBean(HelloService.class);
  hello.sayHello("world !");
 }
}
```

控制台打印出结果如下：

```
how do you do !
how are you !
hello world !
```

正如我们预测的一样，改变了 order 的顺序，切面的执行顺序也发生了改变。

方式二：aspect 使用 @Order 注解

改造切面类，使用注解的方式，不用实现 Ordered 了。

日志切面1：

```
package com.davidwang456.test;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import org.springframework.core.annotation.Order;

@Aspect
@Order(5)
public class LogAspect{
 @Before("execution(* sayHello(..))")
 public void beforeHello() {
  System.out.println("how are you !");
 }
}
```

级别为 5。

日志切面 2，日志级别为 10：

```
package com.davidwang456.test;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import org.springframework.core.annotation.Order;

@Aspect
@Order(10)
public class LogAspect2{
 @Before("execution(* sayHello(..))")
 public void beforeHello() {
  System.out.println("how do you do !");
 }
}
```

运行上面的测试程序，控制台打印结果如下：

```
how are you !
how do you do !
hello world !
```

Aspect 排序原理

AspectJProxyFactory 硬编码方式

测试程序中 AspectJProxyFactory 增加切面类，其 debug 内部如下：

```
/**
 * Add an aspect of the supplied type to the end of the advice chain.
 * @param aspectClass the AspectJ aspect class
 */
public void addAspect(Class<?> aspectClass) {
 String aspectName = aspectClass.getName();
 AspectMetadata am = createAspectMetadata(aspectClass, aspectName);
 MetadataAwareAspectInstanceFactory instanceFactory = createAspectInstanceFactory(am, aspectClass, aspectName);
 addAdvisorsFromAspectInstanceFactory(instanceFactory);
}


/**
 * Add all {@link Advisor Advisors} from the supplied {@link MetadataAwareAspectInstanceFactory}
 * to the current chain. Exposes any special purpose {@link Advisor Advisors} if needed.
 * @see AspectJProxyUtils#makeAdvisorChainAspectJCapableIfNecessary(List)
 */
private void addAdvisorsFromAspectInstanceFactory(MetadataAwareAspectInstanceFactory instanceFactory) {
 List<Advisor> advisors = this.aspectFactory.getAdvisors(instanceFactory);
 Class<?> targetClass = getTargetClass();
 Assert.state(targetClass != null, "Unresolvable target class");
 advisors = AopUtils.findAdvisorsThatCanApply(advisors, targetClass);
 AspectJProxyUtils.makeAdvisorChainAspectJCapableIfNecessary(advisors);
 AnnotationAwareOrderComparator.sort(advisors);
 addAdvisors(advisors);
}
```

其中，`AnnotationAwareOrderComparator.sort(advisors);`会将生成的 Advisor 进行排序。使用的排序器为 AnnotationAwareOrderComparator，它会读取 order 的级别进行排序。

xml配置方式

通过以前的章节，我们知道不管使用何种动态代理方式，代理的生成实现在 DefaultAopProxyFactory 中，那么在此类中打印出调用链路，可以找出脉络。DefaultAopProxyFactory.java

```
@Override
public AopProxy createAopProxy(AdvisedSupport config) throws AopConfigException {
      //打印调用链路
 StackUtils.getStack();
 if (config.isOptimize() || config.isProxyTargetClass() || hasNoUserSuppliedProxyInterfaces(config)) {
  Class<?> targetClass = config.getTargetClass();
  if (targetClass == null) {
   throw new AopConfigException("TargetSource cannot determine target class: " +
     "Either an interface or a target is required for proxy creation.");
  }
  if (targetClass.isInterface() || Proxy.isProxyClass(targetClass)) {
   return new JdkDynamicAopProxy(config);
  }
  return new ObjenesisCglibAopProxy(config);
 }
 else {
  return new JdkDynamicAopProxy(config);
 }
}
```

打印调用链路的程序如下：

```
package com.davidwang456.test;

public class StackUtils {
 public static void getStack() {
    java.util.Map<Thread, StackTraceElement[]> ts = Thread.getAllStackTraces();
     StackTraceElement[] ste = ts.get(Thread.currentThread());
     int cnt=1;
     for (int i=ste.length-1;i>0;i--) {
      StackTraceElement s=ste[i];
      System.out.println("调用序号："+cnt+" 调用类和方法 "+s.getClassName()+"$"+s.getMethodName());
      cnt++;
     }
 }
}
```

运行测试程序，打印出调用链路：

```
调用序号：1 调用类和方法 com.davidwang456.test.AspectTest$main
调用序号：2 调用类和方法 org.springframework.context.support.ClassPathXmlApplicationContext$<init>
调用序号：3 调用类和方法 org.springframework.context.support.ClassPathXmlApplicationContext$<init>
调用序号：4 调用类和方法 org.springframework.context.support.AbstractApplicationContext$refresh
调用序号：5 调用类和方法 org.springframework.context.support.AbstractApplicationContext$finishBeanFactoryInitialization
调用序号：6 调用类和方法 org.springframework.beans.factory.support.DefaultListableBeanFactory$preInstantiateSingletons
调用序号：7 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$getBean
调用序号：8 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$doGetBean
调用序号：9 调用类和方法 org.springframework.beans.factory.support.DefaultSingletonBeanRegistry$getSingleton
调用序号：10 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$$Lambda$9/2036368507$getObject
调用序号：11 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$lambda$0
调用序号：12 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$createBean
调用序号：13 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$doCreateBean
调用序号：14 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$initializeBean
调用序号：15 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$applyBeanPostProcessorsAfterInitializ
ation
调用序号：16 调用类和方法 org.springframework.aop.framework.autoproxy.AbstractAutoProxyCreator$postProcessAfterInitialization
调用序号：17 调用类和方法 org.springframework.aop.framework.autoproxy.AbstractAutoProxyCreator$wrapIfNecessary
调用序号：18 调用类和方法 org.springframework.aop.framework.autoproxy.AbstractAutoProxyCreator$createProxy
调用序号：19 调用类和方法 org.springframework.aop.framework.ProxyFactory$getProxy
调用序号：20 调用类和方法 org.springframework.aop.framework.ProxyCreatorSupport$createAopProxy
调用序号：21 调用类和方法 org.springframework.aop.framework.DefaultAopProxyFactory$createAopProxy
调用序号：22 调用类和方法 com.davidwang456.test.StackUtils$getStack
调用序号：23 调用类和方法 java.lang.Thread$getAllStackTraces
```

其中，调用序号 18 调用类 AbstractAutoProxyCreator 的 getAdvicesAndAdvisorsForBean() 方法调用了其父类：AbstractAdvisorAutoProxyCreator 的 getAdvicesAndAdvisorsForBean 方法，它内部调用 AnnotationAwareOrderComparator 的 sort 方法对生成的 Advisor 进行排序。

```
/**
 * Sort advisors based on ordering. Subclasses may choose to override this
 * method to customize the sorting strategy.
 * @param advisors the source List of Advisors
 * @return the sorted List of Advisors
 * @see org.springframework.core.Ordered
 * @see org.springframework.core.annotation.Order
 * @see org.springframework.core.annotation.AnnotationAwareOrderComparator
 */
protected List<Advisor> sortAdvisors(List<Advisor> advisors) {
 AnnotationAwareOrderComparator.sort(advisors);
 return advisors;
}
```

可见，AnnotationAwareOrderComparator 实现了 OrderComparator 完成 Advisor 的排序。支持接口继承或者注解方式。

总结

通过 Aspect 直接的协调，我们又学习了排序实现的方式：**实现 Ordered 接口或者使用 @Order 注解**

可用于定义的 Aspect 的执行指定执行顺序，值越小，越先执行。

Spring 利用 @Order 控制配置类的加载顺序。

Spring 在加载 Bean 的时候，有用到 order 注解。

}

44 Spring控制器Controller如何设置AOP?

46 Spring AOP 总结及面试热点题目集萃