基于@Aspect注解的aop

在切面类上面需要加注解@Aspect @Componet

在配置上面添加@Configuration @EnableAspectJAutoProxy 这个注解让aspectj支持java配置

声明切点@Pointcut

springaop在切点表达式上支持的aspectj标志符

execution ：匹配方法执行的连接点，用得最多的关键字

within：用指定类型限定连接点的匹配，满足指定的参数类型（多个类型，满足其中之一）才会执行方法

this：连接点的bean引用是给定类型

target：连接点的限定匹配，被代理对象实例是给定的类型

args：参数是给定类型才匹配

@target：被代理对象的类给定类型的注解

@args：运行时传递参数类型具有给定的注解

@within：被代理对象类具有指定注解

@annotation：

用&& ，||， !连接切点表达式

\*

**@Pointcut**("within(com.xyz.myapp.trading..\*)")

匹配trading包下面的所有方法

**@Pointcut**("execution(public \* \*(..))")

匹配所有的公共方法

**@Pointcut**("anyPublicOperation() && inTrading()")

匹配trading包下面的所有公共方法

execution(modifiers-pattern? ret-type-pattern declaring-type-pattern.name-pattern(param-pattern) throws-pattern?)

ret-type-pattern，name-pattern，param-pattern are not optional

ret-type-pattern：连接点必须满足的返回类型，\*通常被用来匹配所有类型。也可以是全限定类型。

name-pattern：匹配方法名。可以用\*和部分使用\*

param-pattern：参数规则要稍微复杂些。()匹配没有参数的方法；(..)匹配0到多个参数的方法；(\*)匹配带有一个任意类型参数的方法；(\*, String)匹配带有2个参数的方法，第一个参数可以是任意类型，第二个参数必须是String类型；

declaring-type-pattern：用.结尾连接name-pattern

\*表示匹配所有的字符

CommonPointcuts

**package** com.xyz.myapp;

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

**import** org.aspectj.lang.annotation.Pointcut;

**@Aspect**

**public** **class** **CommonPointcuts** {

*/\*\**

*\* A join point is in the web layer if the method is defined*

*\* in a type in the com.xyz.myapp.web package or any sub-package*

*\* under that.*

*\*/*

**@Pointcut**("within(com.xyz.myapp.web..\*)")

**public** **void** **inWebLayer**() {}

*/\*\**

*\* A join point is in the service layer if the method is defined*

*\* in a type in the com.xyz.myapp.service package or any sub-package*

*\* under that.*

*\*/*

**@Pointcut**("within(com.xyz.myapp.service..\*)")

**public** **void** **inServiceLayer**() {}

*/\*\**

*\* A join point is in the data access layer if the method is defined*

*\* in a type in the com.xyz.myapp.dao package or any sub-package*

*\* under that.*

*\*/*

**@Pointcut**("within(com.xyz.myapp.dao..\*)")

**public** **void** **inDataAccessLayer**() {}

*/\*\**

*\* A business service is the execution of any method defined on a service*

*\* interface. This definition assumes that interfaces are placed in the*

*\* "service" package, and that implementation types are in sub-packages.*

*\**

*\* If you group service interfaces by functional area (for example,*

*\* in packages com.xyz.myapp.abc.service and com.xyz.myapp.def.service) then*

*\* the pointcut expression "execution(\* com.xyz.myapp..service.\*.\*(..))"*

*\* could be used instead.*

*\**

*\* Alternatively, you can write the expression using the 'bean'*

*\* PCD, like so "bean(\*Service)". (This assumes that you have*

*\* named your Spring service beans in a consistent fashion.)*

*\*/*

**@Pointcut**("execution(\* com.xyz.myapp..service.\*.\*(..))")

**public** **void** **businessService**() {}

*/\*\**

*\* A data access operation is the execution of any method defined on a*

*\* dao interface. This definition assumes that interfaces are placed in the*

*\* "dao" package, and that implementation types are in sub-packages.*

*\*/*

**@Pointcut**("execution(\* com.xyz.myapp.dao.\*.\*(..))")

**public** **void** **dataAccessOperation**() {}

}

任何public方法的执行

execution(public \* \*(..))

任何以set开始的方法

execution(\* set\*(..))

在AcountService接口中定义的任何方法

execution(\* com.xyz.service.AccountService.\*(..))

在service包中定义的任何方法

execution(\* com.xyz.service.\*.\*(..))

在service包和子包中的所有方法

execution(\* com.xyz.service..\*.\*(..))

在service包中的任何连接点

within(com.xyz.service.\*)

在service包和子包中的任何连接点

within(com.xyz.service..\*)

实现了AccountService接口的proxy中的任何连接点

this(com.xyz.service.AccountService)

实现了AccountService接口的target（被代理对象）中的任何连接点

target(com.xyz.service.AccountService)

只有一个参数，且运行时传入参数类型是Serializable。换句话说是实参

args(java.io.Serializable)

execution定义的是方法signature声明一个Serializable类型的参数。 换句话说是形参

execution(\* \*(java.io.Serializable))

target对象（被代理对象）上有@Transactional注解的连接点

@target(org.springframework.transaction.annotation.Transactional)

target对象的声明类型上面有@Transactional注解的连接点 这个好像包括静态方法

@within(org.springframework.transaction.annotation.Transactional)

执行方法上面有@Transactional注解的连接点

@annotation(org.springframework.transaction.annotation.Transactional)

连接点带有一个参数，运行时实参的类型具有@Classified注解

@args(com.xyz.security.Classified)

名字是tradeService的spring bean的连接点

bean(tradeService)

在advice body中怎么access这proxy对象呢？

在advice body中怎么access这target对象呢？

在advice body中怎么access这方法的参数呢？

@Around注解的消息方法第一个参数必须是ProceedingJoinPoint，然后在消息方法体中调用ProceedingJoinPoint的proceed()方法来调用连接点方法（拦截的方法，或者叫被代理的方法）

另一个重载方法proceed(Object[] var1)，var1作为被调用方法的参数。

around通知的proceed(Object[] var1)和proceed()用aspectj编译器有点点差异。对于用传统的Aspectj语言编写的around通知，传递给proceed()方法的参数数量必须和传给消息通知方法的参数数量一致（而不是被代理方法参数的数量）。需要注意用Aspectj编译器和编织器编译用@Aspectj注解针对spring编写的切面 且使用带有参数的proceed的差别。

proceed()在around方法中可以调用1次，多次，或者一次也不掉用，都是可以的

make argument and other contextual values available to advice body

how to write generic advice——that can find out about the method the advice is currently advising

任何消息方法都可以声明JoinPoint类型的参数作为第一个参数。注意：around消息要求第一个参数是ProceeddingJoinPoint类型，他是JoinPoint参数的子类。

JoinPoint接口提供了一系列有用的方法。

* getArgs(): Returns the method arguments.
* getThis(): Returns the proxy object.
* getTarget(): Returns the target object.
* getSignature(): Returns a description of the method that is being advised.
* toString(): Prints a useful description of the method being advised.

传递参数给消息方法

binding form of args。如果使用参数名代替类型名称在args表达式中，当通知的方法被调用相应的参数值被传递作为参数的值。

例如：假设你想通知dao操作的执行，它有一个Acount对象作为第一个参数，你需要在通知方法体里面访问acount。你可以配置如下：

**@Before**("com.xyz.myapp.CommonPointcuts.dataAccessOperation() && args(account,..)")

**public** **void** **validateAccount**(Account account) {

*// ...*

}

切点表达式args(aount, ..)部分有两个作用，1、限制只匹配至少有一个参数的方法执行，并且传递的实参是Acount对象。2、通过advice方法的acount参数来访问Acount对象实参。

另一种书写形式

**@Pointcut**("com.xyz.myapp.CommonPointcuts.dataAccessOperation() && args(account,..)")

**private** **void** **accountDataAccessOperation**(Account account) {}

**@Before**("accountDataAccessOperation(account)")

**public** **void** **validateAccount**(Account account) {

*// ...*

}

this——proxy object

target——target object

annotations（@within， @target， @annotation， @args）

声明注解Auditable

**@Retention**(RetentionPolicy.RUNTIME)

**@Target**(ElementType.METHOD)

**public** **@interface** Auditable {

AuditCode **value**();

}

切点，有@Auditable注解的方法的执行，并将注解传递到advice方法中。

**@Before**("com.xyz.lib.Pointcuts.anyPublicMethod() && @annotation(auditable)")

**public** **void** **audit**(Auditable auditable) {

AuditCode code = auditable.value();

*// ...*

}

消息方法参数和泛型

**public** **interface** **Sample**<**T**> {

**void** **sampleGenericMethod**(T param);

**void** **sampleGenericCollectionMethod**(Collection<T> param);

}

可以通过制定advice方法参数类型，来限定匹配这种类型参数类型的执行。

**@Before**("execution(\* ..Sample+.sampleGenericMethod(\*)) && args(param)")

**public** **void** **beforeSampleMethod**(MyType param) {

*// Advice implementation*

}

下面这种是不行的，不能这样定义pointcut

**@Before**("execution(\* ..Sample+.sampleGenericCollectionMethod(\*)) && args(param)")

**public** **void** **beforeSampleMethod**(Collection<MyType> param) {

*// Advice implementation*

}

To achieve something similar to this, you have to type the parameter to Collection<?> and manually check the type of the elements.

如果参数名称明确指定，指定的参数名称被使用。通知和切点注解都有一个可选属性，它可以用来指定被注解的方法的参数名称

**@Before**(value="com.xyz.lib.Pointcuts.anyPublicMethod() && target(bean) && @annotation(auditable)",

argNames="bean,auditable")

**public** **void** **audit**(Object bean, Auditable auditable) {

AuditCode code = auditable.value();

*// ... use code and bean*

}

如果第一个参数类型是JoinPoint，ProceedingJoinPoint，JoinPoint.StaticPart，在argNames属性中省略

**@Before**(value="com.xyz.lib.Pointcuts.anyPublicMethod() && target(bean) && @annotation(auditable)",

argNames="bean,auditable")

**public** **void** **audit**(JoinPoint jp, Object bean, Auditable auditable) {

AuditCode code = auditable.value();

*// ... use code, bean, and jp*

}

通知执行顺序

on the way in 进来的逻辑 优先级高的执行

target的方法——连接点

on the way out 出去的逻辑 优先级高的后执行

如果没有指定优先级，是没有优先级的。

可以通过切面类implement org.springframework.core.Ordered接口，也可以用@Order注解

配置的order值越小，优先级越高。