对spring的AOP\_AspectJ的学习：

# AspectJ

#### AOP Proxies

Spring AOP defaults to using standard JDK dynamic proxies for AOP proxies. This enables any interface (or set of interfaces) to be proxied.

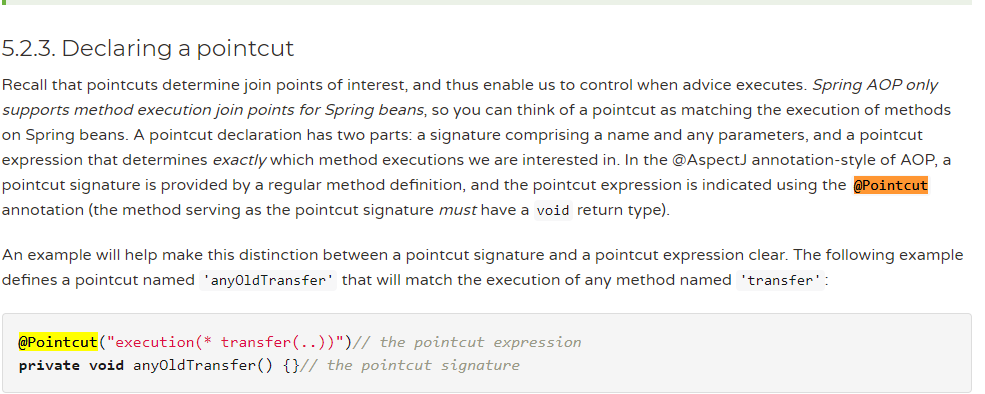
Spring AOP can also use CGLIB proxies. This is necessary to proxy classes rather than interfaces. CGLIB is used by default if a business object does not implement an interface. As it is good practice to program to interfaces rather than classes; business classes normally will implement one or more business interfaces. It is possible to [force the use of CGLIB](https://docs.spring.io/spring/docs/5.0.5.BUILD-SNAPSHOT/spring-framework-reference/core.html#aop-proxying), in those (hopefully rare) cases where you need to advise a method that is not declared on an interface, or where you need to pass a proxied object to a method as a concrete type.

It is important to grasp the fact that Spring AOP is proxy-based. See [Understanding AOP proxies](https://docs.spring.io/spring/docs/5.0.5.BUILD-SNAPSHOT/spring-framework-reference/core.html#aop-understanding-aop-proxies) for a thorough examination of exactly what this implementation detail actually means.

### 5.2. @AspectJ support

@AspectJ refers to a style of declaring aspects as regular Java classes annotated with annotations. The @AspectJ style was introduced by the [AspectJ project](https://www.eclipse.org/aspectj) as part of the AspectJ 5 release. Spring interprets the same annotations as AspectJ 5, using a library supplied by AspectJ for pointcut parsing and matching. The AOP runtime is still pure Spring AOP though, and there is no dependency on the AspectJ compiler or weaver.

## 声明切点pointcut



A pointcut declaration has two parts: a signature comprising a name and any parameters, and a pointcut expression that determines exactly which method executions we are interested in. In the @AspectJ annotation-style of AOP, a pointcut signature is provided by a regular method definition, and the pointcut expression is indicated using the @Pointcut annotation (the method serving as the pointcut signature must have a void return type).

### 切点的构成

切点由俩部分构成：pointcut signature + pointcut expression。

### Pointcut expression的种类：

Some examples of common pointcut expressions are given below.

：

* the execution of any public method:

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

* the execution of any method with a name beginning with "set":

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

* the execution of any method defined by the AccountService interface:

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

* the execution of any method defined in the service package:

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

* any join point (method execution only in Spring AOP) where the proxy implements the AccountService interface:

this(com.xyz.service.AccountService)

* any join point (method execution only in Spring AOP) where the target object has an @Transactional annotation:

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

* any join point (method execution only in Spring AOP) where the executing method has an @Transactional annotation:

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

### 使用@annotation做切点表达式pointcut expression

* any join point (method execution only in Spring AOP) where the executing method has an @Transactional annotation:

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

e.g:

我自定义了一个注解，此时这个注解的作用就是一个Mark标记：

