[AOP概念](https://docs.spring.io/spring-framework/reference/core/aop/introduction-defn.html)

[AOP API](https://docs.spring.io/spring-framework/reference/core/aop-api/pointcuts.html)

切面導向橫切關注點（如日誌、事務等）提取出來

Spring.aop 和Spring.aspectj 功能上aspectj (@Aspect)功能較廣有更多annotation

AOP中的 **切入點（Pointcut）**和 **通知（Advice）介紹**

* 切入點(Pointcut)
* Pointcut介面提供兩個屬性

ClassFilter getClassFilter();

MethodMatcher getMethodMatcher();

* 有許多實作(例如以下)
* StaticMethodMatcherPointcut (abstract class)

只有首次呼叫方法時評估切入點一次,效能上會比較好,但因基於class和method,無法考慮method的傳入參數

實作類別為JdkRegexpMethodPointcut

(pattern可用regex)

* DynamicMethodMatcherPointer (abstract class)

評估成本高於StaticMethodMatcherPointcut,因為還多了method的傳入參數,也就是每次呼叫都要先取得參數值

* ControlFlowPointcut (實作類別)

類似功能較弱的Aspectj的cflow

* [切入點表達式](https://docs.spring.io/spring-framework/reference/core/aop/ataspectj/pointcuts.html)
* execution: 匹配method
* within: 匹配types
* this: 使用Spring AOP proxy時,指定interface或abstract他們的實作或是子類皆會被選定
* target: 匹配目標物件(class)
* args: 匹配傳入參數
* @target: 匹配目標物件(class)上的annotation
* @args:匹配帶有annotation的傳入參數
* @within:匹配帶有annotation的types
* @annotation:匹配帶有annotation的method
* 通知(Advice)
  + 「在何時執行某個功能」
    - @Before：在目標方法執行之前執行。

除非拋出例外否則無法中斷流程道連接點(join point)

* + - @After：在目標方法執行之後執行。

連接點正常結束的返回 或拋出異常地返回,分三種類型

有正常返回Advice的@AfterReturning

或拋出異常Advice的@AfterThrowing

無論正常或異常都會跑的(final)Advice的@After

* + - @Around：包裹目標方法執行，可以控制方法的執行。

使用Joinpoint(介面)的process()方法串接原來連接點

此方法前後皆可以做事情

* + - @AfterReturning：目標方法成功執行後執行。
    - @AfterThrowing：目標方法拋出異常後執行。
  + 可以是Interceptor(許多框架建議做chain of Interceptor)

Spring有提供一個**RegexpMethodPointcutAdvisor**

**上層的抽象類別實作了PointcutAdvisor (含有Pointcut和Advisor介面)**

