Let us begin by defining some central AOP concepts and terminology.

我们首先定义一些AOP核心概念和术语。

These terms are not Spring-specific…

这些条款不是Spring特有的。

unfortunately, AOP terminology is not particularly intuitive;

很不巧，AOP术语不是特别的直观。

however, it would be even more confusing if Spring used its own terminology.

如果Spring使用自有术语会使这些概念更加令人困惑。

*Aspect*:

a modularization of a concern that cuts across multiple classes.

一个模块化的关注点，可能横跨（横切）多个类。

Transaction management is a good example of a crosscutting concern in enterprise Java applications.

事务管理是一个很好的例子，在企业Java应用程序中它是一个横切关注。

In Spring AOP, aspects are implemented using regular classes (the [schema-based approach](http://docs.spring.io/spring/docs/4.0.1.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle/#aop-schema)) or regular classes annotated with the @Aspect annotation (the[@AspectJ style](http://docs.spring.io/spring/docs/4.0.1.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle/#aop-ataspectj)).

在Spring的AOP中，Aspect由常规类或被@Aspect注解的常规类实现。

*Join point*:

a point during the execution of a program, such as the execution of a method or the handling of an exception.

程序执行过程中的一个位点（一个运行时事件），例如某个方法的执行、或一个异常的处理。

In Spring AOP, a join point *always* represents a method execution.

在Spring的AOP中，一个Join Point始终代表某个方法的执行

*Advice*:

action taken by an aspect at a particular join point.

一个Aspect在特定的Join Point采取的动作（运行时事件细分）。

Different types of advice include "around," "before" and "after" advice.

Advice包括不同的类型：“around”，“before”和“after”。

(Advice types are discussed below.)

以下讨论不同的Advice类型。

Many AOP frameworks, including Spring, model an advice as an *interceptor*, maintaining a chain of interceptors *around* the join point.

许多AOP框架（包括Spring），把Advice建模为拦截器，在Join Point附近维护一个拦截链（Hook链）。

*Pointcut*:

a predicate that matches join points.

一个Join Point集合的匹配谓词。

Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name).

Advice和一个Pointcut表达式关联在一起，在任何匹配的Join Point运行。

例如，具有特定命名的方法的执行。

The concept of join points as matched by pointcut expressions is central to AOP,

Join Point由Pointcut表达式匹配的概念对于AOP来说非常重要。

and Spring uses the AspectJ pointcut expression language by default.

而且Spring默认使用AspectJ的Pointcut表达式语言。

*Introduction*:

declaring additional methods or fields on behalf of a type.

为某个类型声明附加的方法或字段。

Spring AOP allows you to introduce new interfaces (and a corresponding implementation) to any advised object.

Spring的AOP允许你对任何Advised对象（被Aspect作用的对象）引入新的接口

和一个对应实现。

For example, you could use an introduction to make a bean implement an IsModified interface, to simplify caching.

例如，你可以使用一个Introduction让一个bean实现实现IsModified接口以便简化缓存。

(An introduction is known as an inter-type declaration in the AspectJ community.)

一个Introduction在AspectJ社区叫做一个内部类型声明。

*Target object*:

object being advised by one or more aspects.

被一个或多个Aspect作用的对象。

Also referred to as the *advised* object.

也可用Advised对象表示。

Since Spring AOP is implemented using runtime proxies,

只要Spring的AOP通过运行时代理实现，

this object will always be a proxied object.

这个对象就永远是一个被代理对象。

*AOP proxy*:

an object created by the AOP framework in order to implement the aspect contracts (advise method executions and so on).

一个由AOP框架为了实现Aspect约束（advise方法执行等）而创建的对象。

In the Spring Framework, an AOP proxy will be a JDK dynamic proxy or a CGLIB proxy.

在Spring框架中，一个AOP Proxy是一个JDK动态代理或一个CGLIB代理。

*Weaving*:

linking aspects with other application types or objects to create an advised object.

将Aspect和其他应用程序的类型或对象关联起来，创建一个Advised对象。

This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime.

这可以在编译期（例如使用AspectJ编译器）、加载期间或运行时完成。

Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime.

Spring的AOP像其他纯Java AOP框架，在运行时进行Weaving。

Types of advice:

Advice的类型：

*Before advice*:

Advice that executes before a join point,

Advice在Join Point之前执行，

but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).

但是这种Advice没有能力阻止执行流执行Join Point（除非抛出异常）

*After returning advice*:

Advice to be executed after a join point completes normally:

Advice在Join Point正常完成后执行

for example, if a method returns without throwing an exception.

例如，如果一个方法没有抛出异常并返回。

*After throwing advice*:

Advice to be executed if a method exits by throwing an exception.

方法异常返回时执行的Advice

*After (finally) advice*:

Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).

无视Join Point退出时的情况一定执行的Advice。

*Around advice*:

Advice that surrounds a join point such as a method invocation.

在Join Point附近的Advice，例如方法调用

This is the most powerful kind of advice.

这是最强大的Advice.

Around advice can perform custom behavior before and after the method invocation.

Around Advice可以在方法调用前后进行自定义行为。

It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.

它有责任选择是通过这个Join Point，还是短路一个Advised方法执行（通过返回自有返回值或抛出异常）。

Around advice is the most general kind of advice.

Around Advice是最通用的Advice。

Since Spring AOP, like AspectJ, provides a full range of advice types,

由于Spring的AOP像AspectJ，提供Advice类型的完全支持。

we recommend that you use the least powerful advice type that can implement the required behavior.

我们建议你使用足以实现必须行为的最不强大的Advice类型。

For example, if you need only to update a cache with the return value of a method,

例如，如果你只需要更新方法返回值的缓存。

you are better off implementing an after returning advice than an around advice,

你最好使用After Returning Advice，而不是Around Advice。

although an around advice can accomplish the same thing.

虽然Around Advice也能实现同样的事情。

Using the most specific advice type provides a simpler programming model with less potential for errors.

使用最特殊的Advice类型可以提供一个更简单具有更少错误可能的编程模型。

For example, you do not need to invoke the proceed() method on the JoinPoint used for around advice, and hence cannot fail to invoke it.

例如，你不需要在JoinPoint调用用于Around Advice的proceed方法，因此不会因为没有调用成功而失败。

In Spring 2.0, all advice parameters are statically typed,

在Spring2.0，所有Advice参数是静态类型。

so that you work with advice parameters of the appropriate type

所以可以使用具有恰当类型的Advice参数。

(the type of the return value from a method execution for example)

例如方法执行的返回值类型，

rather than Object arrays.

而不是object数组。

The concept of join points, matched by pointcuts,

Join Point和与之匹配的Pointcut的概念，

is the key to AOP which distinguishes it from older technologies offering only interception.

是AOP区别于过时技术（仅提供拦截器）的关键。

Pointcuts enable advice to be targeted independently of the Object-Oriented hierarchy.

Pointcut使Advice成为可能，Advice可以有针对性用于面向对象分层（？？）

For example, an around advice providing declarative transaction management

can be applied to a set of methods spanning multiple objects (such as all business operations in the service layer).

例如，一个Around Advice提供声明事务管理，这个Advice可用于一组分散在多个对象的一组方法（例如服务层的所有的业务操作）

Aspect [XML]

{

Advice [XML Element – optional]

( Pointcut [regex] : { Join Points… } )

=> { ***Logic*** }

}