1.概念

AOP是Aspect Oriented Programming的缩写，意思是面向方面编程，与OOP(Object Oriented Programming)面向对象编程对等，都是一种编程思想。从OOP角度分析，我们关注业务的处理逻辑，是属于纵向的行为，从AOP角度分析，我们关注对象行为发生时的问题，是属于横向的行为。AOP有以下概念术语：

Aspect（切面）：一个关注点的模块化，这个关注点可能会横切多个对象。事务管理是J2EE应用中一个关于横切关注点的很好的例子。在Spring AOP中，切面可以使用基于模式）或者基于@Aspect注解的方式来实现。

Joinpoint（连接点）：在程序执行过程中某个特定的点，比如某方法调用的时候或者处理异常的时候。在Spring AOP中，一个连接点总是表示一个方法的执行。

Advice（通知）：在切面的某个特定的连接点上执行的动作。其中包括了“around”、“before”和“after”等不同类型的通知（通知的类型将在后面部分进行讨论）。许多AOP框架（包括Spring）都是以拦截器做通知模型，并维护一个以连接点为中心的拦截器链。

Pointcut（切入点）：匹配连接点的断言。通知和一个切入点表达式关联，并在满足这个切入点的连接点上运行（例如，当执行某个特定名称的方法时）。切入点表达式如何和连接点匹配是AOP的核心：Spring缺省使用AspectJ切入点语法。

Introduction（引入）：用来给一个类型声明额外的方法或属性（也被称为连接类型声明（inter-type declaration））。Spring允许引入新的接口（以及一个对应的实现）到任何被代理的对象。例如，你可以使用引入来使一个bean实现IsModified接口，以便简化缓存机制。

Target Object（目标对象）： 被一个或者多个切面所通知的对象。也被称做被通知（advised）对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个被代理（proxied）对象。

Weaving（织入）：把切面连接到其它的应用程序类型或者对象上，并创建一个被通知的对象。这些可以在编译时（例如使用AspectJ编译器），类加载时和运行时完成。Spring和其他纯Java AOP框架一样，在运行时完成织入。

2.通知类型

前置通知（Before advice）：在某连接点之前执行的通知，但这个通知不能阻止连接点之前的执行流程（除非它抛出一个异常）。

后置通知（After returning advice）：在某连接点正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。

异常通知（After throwing advice）：在方法抛出异常退出时执行的通知。

最终通知（After (finally) advice）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

环绕通知（Around Advice）：包围一个连接点的通知，如方法调用。这是最强大的一种通知类型。环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它自己的返回值或抛出异常来结束执行。

3实现方式

静态代理：由程序员创建或特定工具自动生成源代码，再对其编译；

动态代理：即在运行期动态创建代理类；

动态字节码生成：使用动态字节码生成技术实现AOP原理是在运行期间目标字节码加载后，生成目标类的子类，将切面逻辑加入到子类中，所以cglib实现AOP不需要基于接口；

自定义类加载器：在类加载到JVM之前直接修改某些类的方法，并将切入逻辑织入到这个方法里，然后将修改后的字节码文件交给虚拟机运行；

字节码转换：使用Instrument，开发者可以构建一个字节码转换器，在字节码加载前进行转换。

4.应用场景

性能监控：在方法调用前后记录调用时间，方法执行太长或超时报警

缓存代理：缓存某方法的返回值，下次执行该方法时，直接从缓存里获取

软件破解：使用AOP修改软件的验证类的判断逻辑

记录日志：在方法执行前后记录系统日志

工作流系统：工作流系统需要将业务代码和流程引擎代码混合在一起执行，那么我们可以使用AOP将其分离，并动态挂接业务

权限验证：方法执行前验证是否有权限执行当前方法，没有则抛出没有权限执行异常，由业务代码捕捉