AOP介绍

# Spring框架的两大特性

## IoC：Inversion of Control 控制反转

## AOP: Aspect Oriented Programming面向切面编程

# AOP面向切面编程

**AOP**为**Aspect Orient Programming**的缩写，意为：面向**切面**编程（或者叫做面向方面编程）。aspect切面。(OOP面向对象编程。)

**AOP与OOP互为补充**，面向对象编程将程序分解成各个层次的对象，而面向切面编程将程序运行过程分解成各个切面。这样理解：OOP是从静态角度考虑程序结构，而AOP是从动态角度考虑程序运行过程。

AOP是一种编程方式，不仅适用于Java语言。**AspectJ、Spring AOP只是基于Java语言的AOP实现的框架。**其他语言也有自己的AOP实现。

# AOP的核心思想

**不修改源代码，还可以为类增加新的功能和属性。核心机制就是代理。**

Spring AOP中只实现了将**方法调用**作为连接点。比如每个方法之前需要进行权限校验，如果每个方法都去添加这段重复的校验代码，会非常麻烦，所以利用AOP就很简单，定义好切入点即可把校验代码切入到每个指定的方法前面。

# AOP的作用

AOP专门用于处理系统中分布于各个模块（不同方法）中的交叉点的问题，在JavaEE应用中，通常通过AOP来处理一些具有横切性质的系统级服务，如事物管理、安全检查、缓存、对象池管理等。

AOP已经成为一种非常常用的解决方案。

# AOP中的相关术语（很重要）

## 切面（Aspect）：用于组织多个Advice，Advice放在切面中定义。

## 连接点（Joinpoint）：程序执行过程中明确的点，如方法的调用，或者异常的抛出。在Spring AOP中，只支持连接点是方法的调用。

## 切入点（Pointcut）：就是插入了增强处理的连接点。当某个连接点满足切入表达式时，该连接点将被添加增强处理，该连接点就变成了切入点。

## 增强处理（Advice）或通知：AOP框架在特定的切入点执行的增强处理。

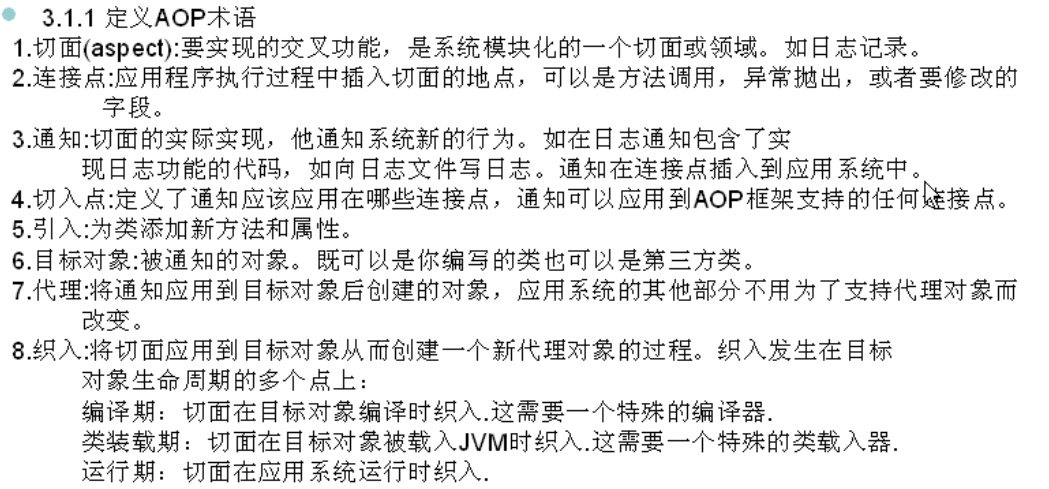
## 引入：将*方法或属性（字段）*添加到被处理的类中。

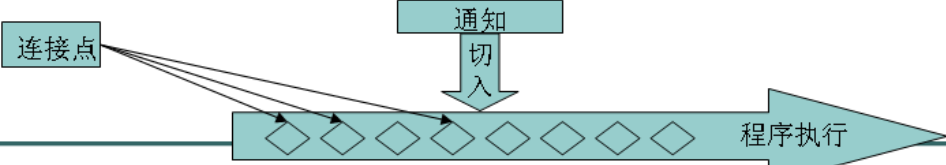
## AOP代理（Proxy）：AOP框架创建的对象，通过代理可以实现对目标对象的增强。Spring AOP的AOP代理可以是JDK动态代理，也可以是cglib代理。

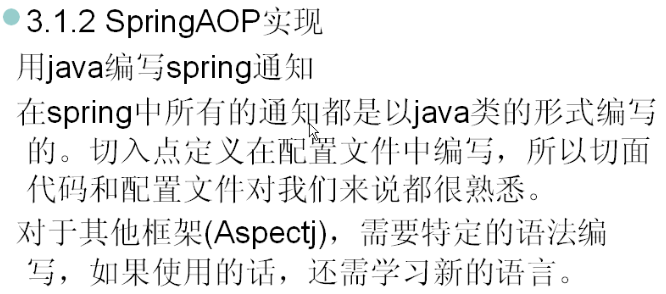
## 织入（Weaving）：将增强处理添加到目标对象中，并创建了一个被增强的对象（AOP代理）的过程就是织入。织入有两种实现方式：一是编译时增强（如AspectJ框架），需要特殊的编译器才能完成；另一个是运行时增强（如Spring AOP）。

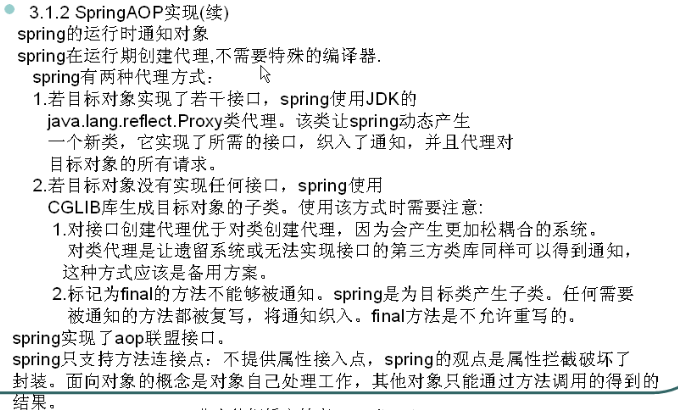
## 目标对象（Target）：被AOP框架增强处理的对象，即被增强的对象。

**代理**：很重要。









# AOP的五种通知（Advice）或为五种增强处理

Before、After、Around、Throws



**引入通知**：可以为类增加新的属性。



# Spring 代理方案：

## JDK的动态代理，接口代理，需要目标类实现接口。

## CGLIB代理：对类代理，备用方案，对final方法不能代理（final方法不允许重写）

# AOP Alliance

AOP联盟

Spring AOP在实现时必须导入**aopalliance.jar**包。

# AOP的实现分类：依据：AOP框架修改源代码的时机

AOP实现分为两类：

## 静态AOP实现

典型代表：AspectJ。

**AOP框架**在**编译阶段**对程序源代码进行修改，即实现对目标类的增强，生成**静态的AOP代理**。（生成的\*.class文件已经被改掉了，使用**特定的编译器**）使用**ajt.bat**对java文件进行编译，ajc.bat是对增强版的**javac.exe**。

编译java文件时： ajc –d \*.java

与javac.exe 使用一样， javac \*.java

ajc与javac 都可以用来编译java文件，区别是ajc.bat命令可以识别AspectJ的语法，而javac.exe无法识别。

## 动态AOP实现

典型代表：SpringAOP。

AOP框架在**运行阶段**动态生成AOP代理（在内存中以**JDK动态代理**或**CGLIB**动态地生成AOP代理类），以实现对目标对象的增强。

SpringAOP就是**动态AOP**的实现的代表，SpringAOP不需要在编译时对目标类进行增强，而是运行时生成目标类的代理类，该代理类要么与目标类实现相同的接口，要么是目标类的子类。总之，代理类都对目标类进行了增强处理，前者是JDK动态代理的处理策略（代理与目标实现相同的接口），后者是cglib代理的处理策略。

# AspectJ介绍

**AspectJ是一个基于Java语言的最完整最流行的AOP框架**，是Java语言的一个AOP实现。在Spring2.0以上的版本中，可以使用**基于AspectJ注解或基于XML配置**的AOP。

AspectJ主要分为两部分：

## 定义了如何表达、定义AOP编程中的语法规范

通过这套语法规范，可以方便地用AOP来解决Java语言中存在的交叉关注点的问题；

## 另一个部分就是工具部分，包括编译器、调试工具等。

单独学习AspectJ ，（不利用IDE），需要安装AspectJ编译器，如

java –jar aspectj-1.8.0.jar

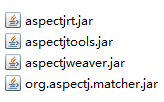
然后配置环境变量，即可利用ajc进行编译java文件了。

aspectj-1.8.0.jar解压后，内部有bin、docs、lib目录。

bin目录下存放aj、aj5、ajc、ajdoc、ajbrowser等命令；其中ajc最常用，类似javac，是对javac的增强，可以用于对普通的java类进行编译时增强。

docs下有很多AspectJ的说应说明、参考手册、API文档等。

lib下有4个jar文件，是AspectJ的核心类库。



Eclipse提供了**AJDT插件**(AspectJ Development Tools)工具。而开发AspectJ可以不依赖与Eclipse。

在利用AspectJ开发AOP时，可以直接创建AuthAspect.java文件：

public aspect AuthAspect{

before() :execution(\* cn.bupt.\*.\*(..)){

}

}

但是，在SpringAOP中无法对该java文件进行编译，所以不能使用这种方式，一般都是使用注解或Xml配置方式开发。

如果单独学习AspectJ，可以查看有关文档。但是目前重点是学会Spring中AOP与IoC的整合，目前没有太多的必要掌握全部的AOP技术。

# SpringAOP与AspectJ的区别

## SpringAOP采用基于代理的AOP实现方案，而AspectJ则采用编译时增强的解决方案。

## SpringAOP框架是动态AOP实现框架的代表，在编译时不对目标类增强，在运行时生成目标类的代理类，要么与目标类实现相同的接口（JDK动态代理），要么是目标类的子类（cglib），通过代理类对目标类进行增强（运行阶段）。

## AspectJ框架是静态AOP实现框架的代表，在编译时对目标类进行增强，需要特定的编译器。

## AspectJ可以单独使用，单独开发，连接点不仅可以是方法调用，还可以是对成员变量的访问和更新。而SpringAOP只能把方法调用作为连接点。

## SpringAOP侧重于和SpringIoC容器之间的整合，用于帮助企业级开发中的问题，并不是要提供完整的AOP实现。

## 从Spring2.0开始，Spring已经无缝地整合SpringAOP、IoC、AspectJ，使得所有的AOP应用完全融入Spring框架中，这样的集成不会影响Spring AOP API或者AOP Alliance API。