一、相关概念

- 1.1 AOP(Aspect Oriented Programming)
  - AOP：面向切面编程
  - 不通过修改源代码的方式，在主干功能中添加新的功能
  - 仍是为了降低业务逻辑各部分之间的==耦合度==
  - 在不惊动(改动)原有设计(代码)的前提下，想给谁添加功能就给谁添加。这个也就是Spring的理念。
- 1.1.1 连接点
  - 类里面哪些方法可以被增强，这些方法称为连接点（理论上可以被增强的方法）
- 1.1.2 切入点
  - 实际真正被增强的方法被，称为切入点（实际上被增强的方法）
  - **切入点表达式标准格式**：动作关键字(访问修饰符 返回值 包名.类/接口名.方法名（参数）异常名)
  - **切入点表达式**
    - `execution(* com.yitsd.aopAnno.User.add(..))`

- > execution(void com.itheima.dao.BookDao.update()) 匹配接口，能匹配到

> execution(void com.itheima.dao.impl.BookDaoImpl.update()) 匹配实现类，能匹配到

> execution(* com.itheima.dao.impl.BookDaoImpl.update()) 返回值任意，能匹配到

> execution(* com.itheima.dao.impl.BookDaoImpl.update(*)) 返回值任意，但是update方法必须要有一个参数，无法匹配，要想匹配需要在update接口和实现类添加参数

> execution(void com.....update()) 返回值为void,com包下的任意包三层包下的任意类的update方法，匹配到的是实现类，能匹配

> execution(void com...*.update()) 返回值为void,com包下的任意两层包下的任意类的update方法，匹配到的是接口，能匹配

> execution(void *..update()) 返回值为void，方法名是update的任意包下的任意类，能匹配

> execution(* ..(..)) 匹配项目中任意类的任意方法，能匹配，但是不建议使用这种方式，影响范围广

> execution(* ..*u*(..)) 匹配项目中任意包任意类下只要以u开头的方法，update方法能满足，能匹配

> execution(* *..*e(..)) 匹配项目中任意包任意类下只要以e结尾的方法，update和save方法能满足，能匹配

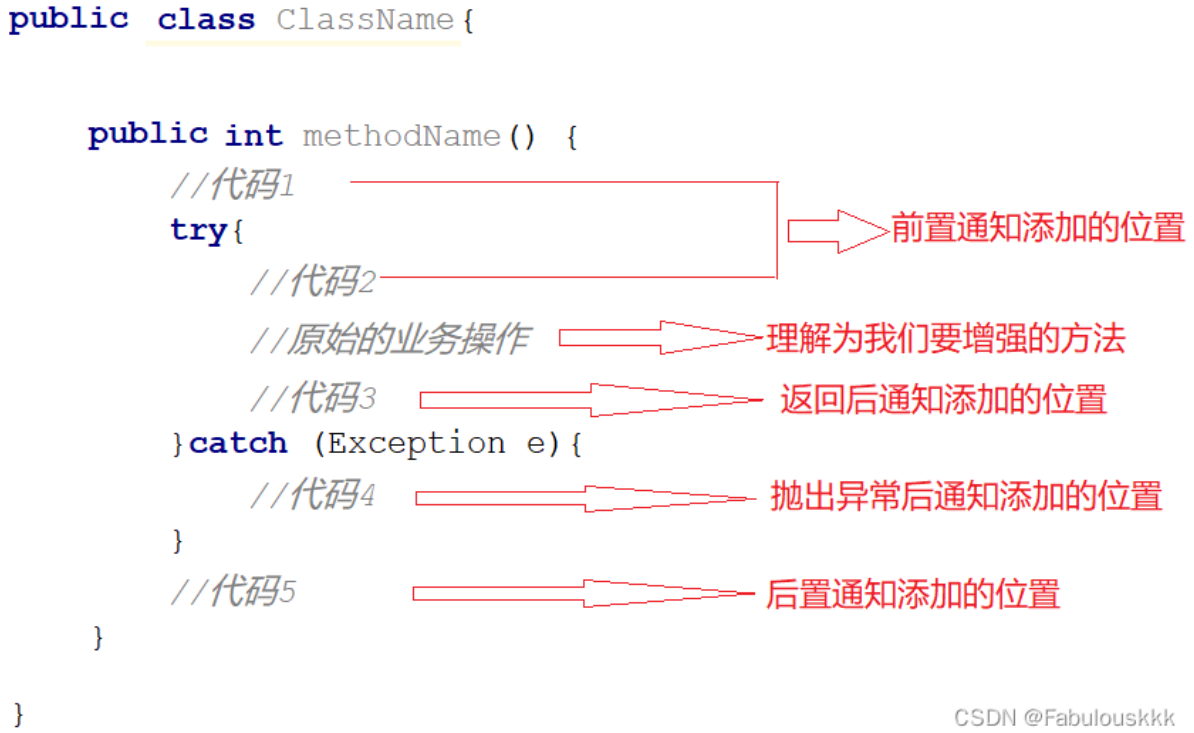
> execution(void com..*()) 返回值为void，com包下的任意包任意类任意方法，能匹配，*代表的是方法

> execution(* com.itheima.*.*Service.find*(..)) 将项目中所有业务层方法的以find开头的方法匹配

> execution(* com.itheima.*.*Service.save*(..)) 将项目中所有业务层方法的以save开头的方法匹配

- 切入点表达式描述通配符：
  - *：匹配任意符号（常用）
  - ..：匹配多个连续的任意符号（常用）
  - +：匹配子类类型
- 切入点表达式书写技巧
  - 按==标准规范==开发
  - 查询操作的返回值建议使用*匹配
  - 减少使用..的形式描述包
  - ==对接口进行描述==，使用*表示模块名，例如UserService的匹配描述为*Service
  - 方法名书写保留动词，例如get，使用*表示名词，例如getById匹配描述为getBy*
  - 参数根据实际情况灵活调整
- 1.1.3 通知（增强）

- 实际增强的逻辑的部分称为通知（增强）



  - `@Before`：追加功能到方法执行前，类似于在代码1或者代码2添加内容
  - `@AfterReturning`：追加功能到方法执行后，只有方法正常执行结束后才进行,类似于在代码3添加内容，如果方法执行抛出异常，返回后通知将不会被添加
  - `@AfterThrowing`：追加功能到方法抛出异常后，只有方法执行出异常才进行,类似于在代码4添加内容，只有方法抛出异常后才会被添加
  - `@After`：类似于try-catch-finally中的==finally==，无论有无异常都会被执行，类似于在代码5添加内容
  - `@Around`：它可以追加功能到方法执行的前后
  - **REMARK**
    - 环绕通知必须依赖形参ProceedingJoinPoint才能实现对原始方法的调用，进而实现原始方法调用前后同时添加通知
    - 通知中如果未使用ProceedingJoinPoint对原始方法进行调用将跳过原始方法的执行

- 1.1.4 切面
  - 切面是一个动作
  - 把通知应用到切入点的过程称为==切面==

二、AOP操作（AspectJ注解）

- 2.1 创建类，在类里面定义方法

```
public class User {
    public void add(){
        System.out.println("add......");
```

```
    }
}
```

- 2.2 创建增强类（编写增强逻辑）
    - 在增强类里面，创建方法，让不同的方法代表不同的通知类型

- 2.3 进行通知的配置

1. 开启注解扫描（SpringConfig）
2. 创建User（被增强类）和UserProxy（增强类）的对象（注解）==[proxy->代理]==
3. 在增强类上添加注解`@Aspect`
4. 在被增强的类上添加注解`@EnableAspectJAutoProxy`
5. 配置不同类型的通知
    - 在增强类的里面，在通知方法上添加通知类型注解，使用切入点表达式配置
6. 有多个增强类对同一个方法进行增强，设置增强类型优先级
    - 在增强类的上方添加注解`@Order(数字类型的值)`,数字越小，优先级越高