- 反射
  - 理解
    - 程序——>javac.exe运行生成.class运行时类——>java.exe对运行时类进行解析
    - 意义：获取.class运行时类，并把它作为Class类的一个实例化对象。获取它的属性、方法、构造器
    - 动态性：当我们并不知道要实例化什么类的对象，通过反射依具体情况具体创建
  - 创建
    - 哪个类创建了这个对象：获取该运行类作为实例的四种方法
    - 实现
      - 调用运行时类(.class)属性：Class cp1 = Person.class;
      - getClass方法
        - Person p1 = new Person()
        - Class cp2 = p1.getClass();
      - 常用法（面越广越灵活越常用）：调用Class类的静态方法:Class cp3 = Class.forName("String的类文件地址")
      - 使用类的加载器ClassLoder
    - 方法
      - getConstructors()，获取该类所有的public构造器
      - getDeclaredConstructors(),获取所有的构造器
      - 获取属性（Field）、方法、修饰符、文件、泛型、实现类：都是顺便获取父类的，含declare的是只获取本身的
      - 获取：更重要
        - 运行时类的父类
        - 运行时类的带泛型的父类
        - 运行时类的带泛型的父类的泛型
        - 运行时类实现的接口
  - 类的加载
    - 将class文件字节码内容加载到内存中，生成一个代表这个类的java.lang.Class对象
    - 配置文件的加载
      - 创建配置文件的实例化对象：Properties p = new Properties()
      - 获取加载器：ClassLoader c = Test.class.getClassLoader()
      - 获取数据流：InputStream is= c.getResourceAsStream("文件路径")
      - 加载：p.load(is)
  - 获取运行时类的特定结构
    - 获取属性
      - 1.获取该运行时类的Class：Class clazz = Person.class;
      - 2.获取该运行时类的抽象属性：Field age = clazz.getDeclaredField("age");
      - 注意：必须保证该类是可以改变的：age.setAccessible(true)
      - 3.创建该运行时类的实例对象：Person p = new Person
      - 4.设置该对象的抽象属性：clazz.set(p,31)
      - 5.获取该对象的具体值：clazz.get(p)
    - 获取方法
      - 1.获取方法：Method show = clazz.getDeclaredMethod("show",String.class)
      - 2.设置可访问：show.setAccessible(true)
      - 3.调用 Object o = show.invoke(p3),返回方法返回的值，若本身void则返回nul
  - 代理
    - 理解代理：找个傀儡，每一件想做的事情都用同名方法的障眼法让傀儡去实现，而自己不用出面。
    - 静态代理
      - 都实现同一个父类，实现他们的代理类和被代理类具有同名方法。
      - 1.创建代理类，将被代理类作为构造器的参数传入，在实现方法的时候相当于递归调用成了被代理类的方法实现。
      - 2.创建被代理类，也就是说实际是一一对应的关系
      - 3.实例化两类，调用代理类的方法，相当于以代理类实现了被代理类的功能。
      - 意义：每一个代理类都需要创建一个被代理类，太累了，何不实现代理类的生产工厂，被代理类要一个相应的就给他造一个代理类
    - 动态代理
      - 创建关系框架
        - 1.创建父类接口
        - 2.创建被代理类实现父类接口
        - 3.创建代理类的生产工类厂
          - public static Object getProxyInstance(Object obj){ }
          - 1.里面要获取被代理类的类加载器、实现接口、内部方法，返回代理类
          - Proxy.newProxyInstance(obj.getClass().getClassLoader(),obj.getClass().getInterfaces(),InvocationHandler);
          - 2.实现InvocationHandler接口,重写invoke实现同名方法调用
          - class MyInvocation implements InvocationHandler{ @ override}
          - 需要借助折衷的方式obj传入被代理类
          - public Object invoke(Object o, Method method, Object[] objects) □ return method.invoke(obj, objects);}
        - 实现
          - 1.创建被代理类的实例对象：Bookbuyer b1 = new Bookbuyer();
          - 2.利用加工厂创建代理类:
Book bookshop = (Book) ProxyFactory.getProxyInstance(b1)
          - 3.代理类实现被代理类的对象:bookeshop.read()
      - 问题一　如何解决，根据被代理类，需要创建一个实现同一个接口的代理类？
      - 问题二　如何解决代理类实现调用被代理类的同名方法？
    - 意义
      - 侵入性：需要增强原来代码的功能，无论是实现接口、继承，都需要修改原来的代码
      - 接口耦合AOP，用反射将你映射出来，实现你的功能但是我不修改你原来的代码，实现功能的增强与扩展。这就是代理模式