1. classloader的类加载机制
   1. 动态加载机制
      1. 并非字节码文件运行前一次性加载所有需要的类
      2. 当需要它加载的时候才加载类
   2. static语句块只加载一次
      1. 还记得jvm中类加载的初始化阶段吗，把所有的静态语句块放到一起形成<cinit>方法，并执行它，只要类不卸载，就只加载一次，所以<cinit>方法只执行一次，static语句块也只执行一次。
2. Classloader分类
   1. BootstrapClassLoader
      1. 定义：启动类加载器
      2. 特点：
         1. 顶层类加载器，无法通过getClassLoader方法获取
         2. 加载rt.jar（String、Long、Object啊什么的都在里面）
         3. 用native language编写（C/C++），可以用来加载其他的低层次的类加载器
   2. ExtentionClassLoader
      1. 定义：扩展类加载器
   3. ApplicationClassLoader
      1. 定义：应用类加载器
      2. 特点：
         1. 加载我们自己写的类
   4. 其他
      1. UrlClassLoader等。。。
3. 反射机制常用方法
   1. Class.forName（String name）
      1. 传入一个类的name，通过这个类的name加载这个类，返回这个类的java.lang.class对象
   2. class.newInstance
      1. 返回这个class类的一个实例，返回值为Object
   3. Class.getMethods
      1. 返回这个class的所有public方法的Method对象
   4. Method.invoke（Object，param）
      1. Obejct参数为刚才用newInstance对象创建的实例，有实例才能获取方法，方法又不一定的静态的，所以没实例获取不了方法
      2. Param参数的个数是可变的
4. 反射机制的基础应用
   1. 框架配置文件
      1. 比如上述的Class.forName的name，我不从配置文件中获取，不用反射调用该类的方法，你在代码重构的时候就要修改所有的代码。而如果有反射机制，有配置文件，你从配置文件里读一个name，方法的调用策略根本不用改，因为反射机制完全可以获取到这个类里各种各样的信息，然后根据这些信息去采取不同的策略。
   2. 动态代理
      1. 举个栗子，有一个订单类，订单要打折吧，打折得分类吧，周一的折扣，周二的折扣，周三的折扣。那你要把折扣信息封装在订单累里吗？不，耦合性太高！那用静态代理机制吧！但是要为每一个中不同的打折方式都封装一个代理类，那得多少个代理类，所以，用反射机制+动态代理，只需要获取类的名字，就可以获取类的全部信息，你完全可以在动态代理类的构造函数或者set方法传入一个string，即类的名字，用它来获取折扣方式类的信息，这样就这只要有一个动态代理类就可以了。