**认识Class类：**

1. 要想完成反射操作，则需要使用Class类，Class类是Java反射机制的源头。
2. 在Java中Object类是一切类的父类，对于反射来讲，所有类的对象实际上都是 java.lang.Class类的实例，所有的对象都可以转变为java.lang.Class类型。
3. Class类也应用了泛型技术。
4. Class类表示一个类的本身，通过Class类可以得到一个类的完整结构，如类中 的方法、属性、构造方法、父类、父接口等。
5. 与反射操作有关的类定义在java.lang.reflect包中。

**Class对象的三种实例化方式：** Class本身没有定义任何的构造方法，如果想要实例化Class类对象，则必须通过以下方 法进行实例化。

1. 调用Class类中的静态方法forName( )实例化，并传入类路径；
2. 通过“类.class”进行实例化；
3. 通过“对象.getClass( )”方法实例化。

对于以上三种Class类对象的实例化方式，forName( )最为标准。

**使用Class类实例化对象：**

Class类在开发中最常见的用法就是实例化对象的操作，即通过一个给定的“包.类”路 径来实例化该类对象。使用Class类实例化对象分为调用无参实例化对象和调用有参实 例化对象。

1. 可以使用newInstance( )方法来实例化对象，但必须保证类中存在一个无参构造方法。
2. 如果类中没有无参构造方法，则需要调用Class类中getConstructors( )取得本类的全 部构造方法，然后根据需要调用相应的构造方法并传入参数完成实例化化操作。但 需要注意有参构造的在类中的排列。

对于一个设计好的类来讲，类中应该存在一个无参构造方法。

**使用反射取得类的结构：**

使用反射可以取得一个类的完整结构、包括类实现的全部接口、类继承的父类、类中的 全部构造方法、类中全部的方法、类中全部的属性。

**取得类实现的全部接口**

要想取得一个类所实现的全部接口，则必须使用Class类中getInterface( )方法，此方法 返回一个Class类对象数组，因为，一个类可以有多个接口。

然后返回的Class类对象数组分别调用Class类中的getName( )方法取得接口的名称。

**取得类继承的父类：**

要想取得一个类所继承的父类，则必须使用Class类中getSuperclass( )方法，此方法返 回一个Class类对象，因为一个类只能有一个父类。然后通过返回的Class对象调用 getName( )方法取得父类的名称。

对于没有明确继承父类的类，默认继承Object类。

**取得类中的全部构造方法：**

要想取得一个类中的全部构造方法，则必须使用Class类中getConstructors( )方法，此 方法返回一个Construction类的对象数组，每一个Construction对象就表示一个构造方 法。然后通过直接输出每一个Construction对象取得每一个构造方法。

用户也可以手工拼凑出信息。

**取得类中的全部方法：**

要想取得一个类的全部方法，则必须使用Class类中的getMethods( )方法，此方法返回 一个Method类的对象数组，每一个Method类对象就表示一个方法。然后通过直接输 出每一个Method对象取得每一个方法。并且从父类中的继承而来的方法也会被输出。

用户也可以手工拼凑出信息。

**取的类中的全部属性：**

要想取得一个类中的全部属性，则必须使用Class类中getFields( )方法或者 getDeclaredField( )方法。

1. getFields( )返回一个Field类数组，用来取得实现的接口或父类中的全部的公共属性。
2. getDeclaredFields( )返回一个Field数组，用来取得本类中的全部属性。

返回的Field对象数组，每个Field对象就表示一个属性。然后通过直接输出每一个对象 数组得到属性信息。

用户也可以手工拼凑出信息。

**反射机制的深入应用：**

通过反射还可以调用类中的方法，操作类中属性，调用类中的构造方法，调用类中的 setter和getter方法此，操作数组。

**使用反射调用类中的方法：**

如果想要使用反射调用类中的方法，则需要通过Method类来完成，其操作步骤如下：

1. 通过Class类对象调用getMethod( )方法取得指一个Method类的实例，在调用 getMethod( )方法时，传入需要调用的方法名称，如果此方法有参数的话，还需要指定 参数的类型。
2. 取得Method类实例后，通过此Method实例调用Method类中的invoke( )方法，并 向此方法中传入该类实例（指该Method类所属的类），如果有参数，还需要传入参 数。如果返回值的话，还需要使用变量来接收返回值。

**调用setter及getter方法：**

通过反射调用类中setter和getter方法的实质上也是通过invoke( )方法进行调用。

**通过反射操作属性：**

要想通过反射操作一个类的属性，则需要使用Field类，其步骤如下：

1. 通过Class对象调用getDeclaredField( )方法取得指定的属性，在调用 getDeclaredField( )方法时需要传入属性名称，然后会根据名称返回一个Field类的 实例。
2. 取得相应的Field实例后，因为属性都设置成私有的访问权限，所以需要通过得到的 Filed类实例调用setAccessible(true)方法将其代表的属性设置为能可被外部访问。
3. 然后通过Field实例调用set( )方法为一个对象设置该属性的内容
4. 最后可以通过Filed类实例调用get( )方法得到某对象的属性。

**类的生命周期：**

如果要使用一个类，肯定离不开JVM。在程序执行中JVM通过装载、链接、初始化3 个步骤完成使用。

1. 类的装载是指通过类加载器把class二进制文件装入JVM的方法区，并在堆区创建 描述该类的java.lang.Class对象，用来封装数据。且同一个类只会被JVM加载一次。
2. 链接就是把二进制数据组装成可以运行的状态，链接分为校验、准备和解析3个步 骤。

·校验用来确认此二进制文件是否适合当前的JVM版本

·准备用来为静态成员分配内存空间，并设置默认值

·解析指的是常量池的代码引用转化为直接引用的过程，直到所有的符号引 用都可被运行程序使用。

1. 链接完成后，类可以初始化了，然后就可以正常使用并实例化对象。当没有任何引 用指向Class对象时，将结束类的生命。

**反射工厂设计模式：** 对于简单的工厂设计模式依然存在问题，那就是在每增加一个子类时都需要修改工厂类， 而此时可以通过反射机制来改善工厂类，这样在增加子类时就不需要修改工厂类了。

**结合属性文件的工厂模式：**

将反射应用在工厂设计模式上，这样在操作的时候只需要传入完整的包类名称，但是用 户无法知道一个接口有多少个可以使用的子类，所以此时可以通过属性文件的形式配置 索要的子类信息。

**Java类加载器：**

在Java中主要有三种类加载器：

1. Bootstrap ClassLoader：此加载器采用C++编写，一般开发中是看不到的。
2. Extension ClassLoader：用来进行扩展类的加载，一般对应的是“jre/lib/ext”目录中 的类。
3. AppClassLoader：加载classpath指定的类，是最常用的一种加载器。

通过调用Class类中的getClassLoader( )可以取得一个类的加载器。还可以继续通过 getParent( )得到父类加载器。

**ClassLoader类：**

利用Class类可以通过反射对一个类进行操作，但是Class类只能够加载CLASSPATH中定 义的类文件，这样很大程度上就会带来操作的局限性。为了解决这样的问题，在Java 中又提供了ClassLoader类，ClassLoader类的主要功能是可以由用户自己设置类的加载 路径，例如通过文件加载或者网络加载。

1. 用户可以定义属于自己的类加载器，如果想要定义属于自己的类加载器，那么可以 直接继承ClassLoader类完成。
2. ClassLoader类是一个抽象类，但此类中并没有抽象方法，而是一些普通方法。这样 的目的是为了让用户通过继承使用类中的方法。

**ClassLoader的双亲加载机制：**

对于自己定义的String类，是不能被加载的。因为ClassLoader类对于类的加载使用了 双亲加载机制，也就是说系统类库依然使用系统的类加载器，而只有非系统类库才可以 使用自定义的ClassLoader。String类是系统类的名字，这个类只能被系统类加载器控制， 用户无法定义与之一样的类来实现加载。

**动态代理：**

之前所学的代理设计，属于静态代理，每一个代理类只能为一个接口服务，这样在程序 中必然会产生过多的代理。而使用动态代理只需要通过一个代理类就可完成全部的代理 功能。

在Java中要想实现动态代理，则需要java.lang.reflect.InvocationHandler接口和 java.lang.reflect.Proxy类的支持。

1. 在InvocationHandler接口中只定义了一个抽象方法，此方法有3个参数：

·Object proxy：被代理的对象

·Method mehtod：要调用的方法

·Object[] args：方法调用时所需要的参数，

1. Proxy类是专门完成代理的操作类，可以通过此类作为一个或多个接口动态的生成实 现类。在Proxy类中有一个newProxyInstance( )方法，通过调用此方法可以动态的生 成实现类，此方法有三个参数：

·ClassLoader loader：类加载器

·Class<?>[] interface：得到全部的接口

·InvocationHandler h：得到InvocationHandler接口的子类实例。

完成动态代理，首先需要定义一个InvocationHandler接口的子类，并在此类中编写一个 方法用于绑定对象和生成动态代理类，还需要覆写InvocationHandler接口中invoke方 法，在此方法中完成具体的方法调用。