1. 代理的意义

当我们想要构造一个类的对象时，这个类实现了一个或者多个接口，在编译时可能是不知道这些接口到底是什么。普通的类可以使用newInstance方法或者使用反射找出构造器，但是接口是无法直接实例化的，需要在运行的程序中定义一个新的实现类。为了解决这个问题，代理机制出现了，代理类可以在运行时创建一个全新的类，这样的代理类能够实现你指定的接口。

1. 调用处理器

调用处理器的作用是在调用代理对象的方法时，都会转向调用处理器，调用其中的invoke方法。调用处理器是一个实现了InvocationHandler接口的类对象，这个接口只有一个invoke方法：

Object invoke(Object proxy, Method method, Object[] args)

1. 创建代理对象

要想创建一个代理对象，需要使用Proxy类的newProxyInstance方法，这个方法有三个参数：

类加载器

Class对象数组

调用处理器

Object proxy = Proxy.*newProxyInstance*(ClassLoader.*getSystemClassLoader*(), new Class[]{Comparable.class}, handler);

1. 使用代理的场景
2. 将方法调用路由到远程服务器
3. 在运行的程序中将用户界面时间和动作关联起来
4. 为了调试，跟踪方法调用
5. 调用代理对象的方法

使用代理对象跟踪一个二分查找，首先在elements数组中添加代理，然后调用Arrays类的binarySearch方法在数组中查找一个随机整数，在binarySearch方法中就会有以下的调用：

if(elements[i].compareTo(key)<0)…

由于elements数组中填充了代理对象，所以compareTo调用了调用处理器中的invoke方法，可以看出，只要调用了代理对象的方法，都会调用invoke方法。

1. 代理类的特性

代理类是在程序运行过程中动态创建的，一旦被创建，它们就变成了常规类，与虚拟机中的其他类没有什么区别。所有的代理类都会扩展Proxy类，且一个代理类中只有一个实例字段，就是调用处理器，完成代理对象任务所需要的所有额外数据都需要存储在调用处理器之中。

所有的代理类都会覆盖Object类的toString, equals和hashCode方法，这些方法都会调用处理器的invoke方法。代理类是没有定义名字的，虚拟机将会给它自动生成一个以字符串 $Proxy开头的类名。

使用同一个类加载器和预设的接口，只能得到一个代理类。如果使用同一个类加载器和接口数组调用两次newProxyInstance方法， 将得到同一个类的两个对象。使用Proxy类的isProxyClass方法可以检测出一个特定的Class对象是否表示一个代理类。

1. 代理实例
2. 调用处理器

package proxy;  
  
import java.lang.reflect.InvocationHandler;  
import java.lang.reflect.Method;  
  
public class TraceHandler implements InvocationHandler {  
  
 private Object target;  
 public TraceHandler(Object t){  
 target = t;  
 }  
  
 @Override  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 System.*out*.println(target);  
 System.*out*.println("." + method.getName() + "(");  
 if(args != null){  
 for(int i = 0; i < args.length; ++i){  
 System.*out*.println(args[i]);  
 if(i < args.length-1){  
 System.*out*.println(", ");  
 }  
 }  
 }  
 System.*out*.println(")");  
 return method.invoke(target, args);  
 }  
}

1. 创建代理对象并且调用

package proxy;  
  
import java.lang.reflect.Proxy;  
import java.util.Arrays;  
import java.util.Random;  
  
public class ProxyTest {  
 public static void main(String[] args) {  
 Object[] elements = new Object[1000];  
 for(int i = 0; i < elements.length; ++i){  
 Integer value = i+1;  
 TraceHandler handler = new TraceHandler(value);  
 Object proxy = Proxy.*newProxyInstance*(ClassLoader.*getSystemClassLoader*(), new Class[]{Comparable.class}, handler);  
 elements[i] = proxy;  
 }  
 Integer key = new Random().nextInt(elements.length)+1;  
 int result = Arrays.*binarySearch*(elements, key);  
 if(result >= 0){  
 System.*out*.println(elements[result]);  
 }  
 }  
}