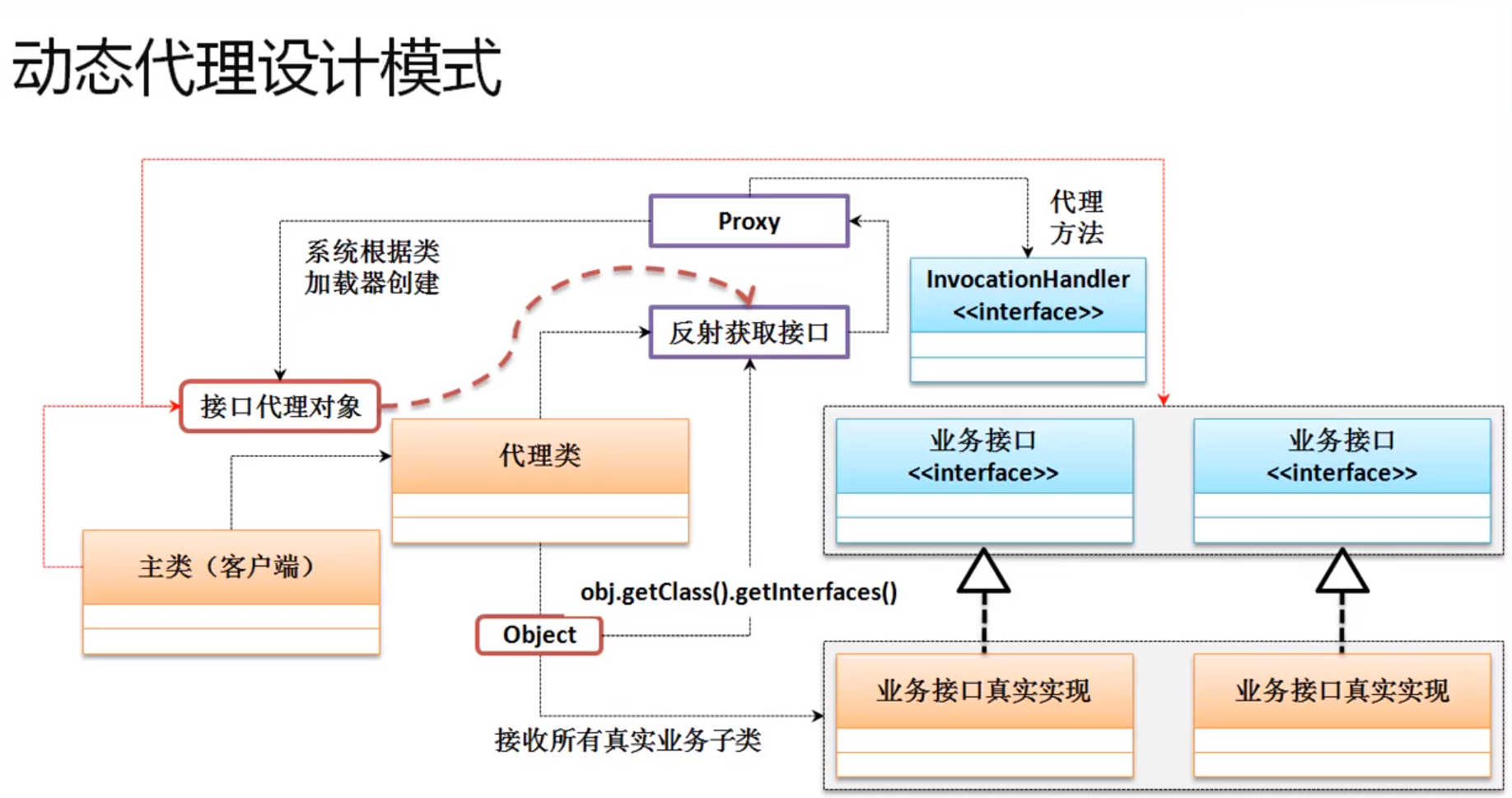
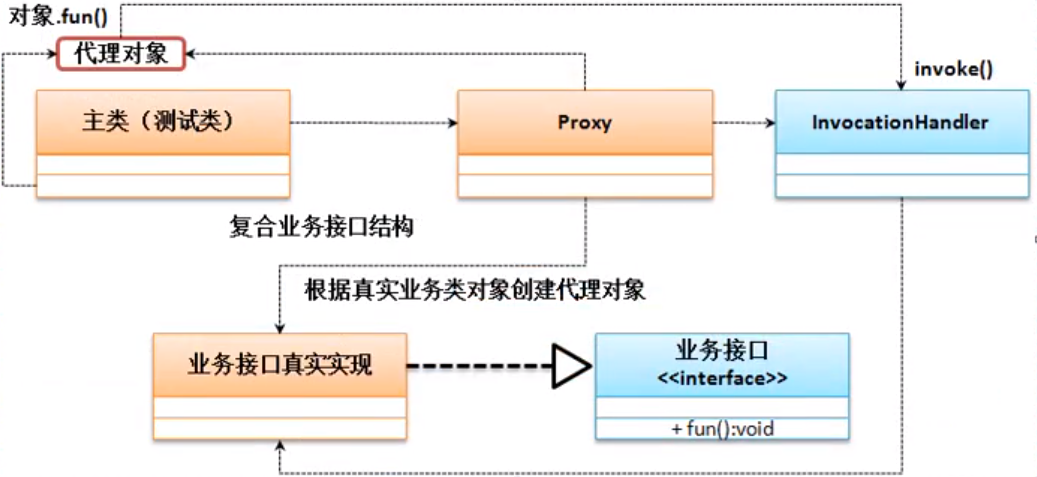
## 动态代理设计模式





### 官方给定的基于接口实现的动态代理

主要是使用到两个类java.lang.reflect.Proxy（负责生成代理类）;java.lang.reflect.InvocationHandler（负责业务对象方法的invoke）;

1. 需要在代理类中绑定实际的业务对象

Object bind(Object target);

该方法需要调用Proxy.*newProxyInstance(业务对象的类加载器，业务对象的接口，以及InvacationHandler对象)*

1. 重写InvocationHandler的invoke方法，封装代理类的公共逻辑和实际业务对象的业务逻辑。
2. 调用接口方法。

|  |
| --- |
| **package** com.pcq.react.proxypattern;  **import** com.pcq.inter.IChannalService;   **import** java.lang.reflect.InvocationHandler; **import** java.lang.reflect.Method; **import** java.lang.reflect.Proxy;  **public class** MyInvocationHandler **implements** InvocationHandler, IChannalService {  **private** Object **target**;   **public** Object bind(Object target) {  **this**.**target** = target;  *//通过第二个参数可见，官方实现的代理是基于接口来实现的* **return** Proxy.*newProxyInstance*(target.getClass().getClassLoader(), target.getClass().getInterfaces(), **this**);  }  @Override  **public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {  Object obj = **null**;  **if**(**this**.isConnect()) {  obj = method.invoke(**target**, args);  **this**.close();  }  **return** obj;  }   @Override  **public boolean** isConnect() {  System.***out***.println(**"build connect......."**);  **return true**;  }   @Override  **public void** close() {  System.***out***.println(**"close connect........"**);  } } |

### 通过第三方开发包CGLIB基于类的动态代理