假设我们有如下的代码

```
public class SecurityServiceProxy
{
    public string Encrypt(string input)
    {
        try
        {
            return new SecurityServiceClient().Encrypt(input);
        }
        catch(Exception ex)
        {
            //Log Exception
            throw;
        }
    }

    public string Decrypt(string input)
    {
        try
        {
            return new SecurityServiceClient().Decrypt(input);
        }
        catch(Exception ex)
        {
            //Log Exception
            throw;
        }
    }
}
```

这个代码中有两段重复（异常处理块重复以及在异常时记Log的逻辑重复），这个重复从AOP的角度来看的话应该是属于横切关注点，所以如果能以AOP的方式解决那应该是最好的方案了，以下实现了一种特殊形式的AOP，代码如下：

```
public class SecurityServiceProxy
{
    public string Encrypt(string input)
    {
        return new SecurityServiceClient().ExecAndLogExcetion(client => client .Encrypt(
    }

    public string Decrypt(string input)
    {
        return new SecurityServiceClient().ExecAndLogExcetion(client => client .Decrypt(
```

```
    }
}
```

现在的代码已经简捷多了，这主要是因为利用到了如下的扩展方法：

```
public static class ObjectExtensions
{
    public static void LockExec<T>(this T obj, Action<T> action) where T : class
    {
        lock (obj)
        {
            action(obj);
        }
    }

    public static void ExecAndLogExcetion<T>(this T obj, Action<T> action)
    {
        try
        {
            action(obj);
        }
        catch (Exception)
        {
            //log excetion
            throw;
        }
    }

    public static void ExecAndLogExcetion<T1, T2>(this T1 obj, T2 obj1, Action<T1, T2>
    {
        try
        {
            action(obj, obj1);
        }
        catch (Exception)
        {
            //log excetion
            throw;
        }
    }

    public static TResult ExecAndLogExcetion<T, TResult>(this T obj, Func<T, TResult>
    {
        try
        {
            return func(obj);
```

```
            }
            catch (Exception)
            {
                //log excetion
                throw;
            }
        }

        public static TResult ExecAndLogExcetion<T1, T2, TResult>(this T1 obj, T2 obj1, Fu
        {
            try
            {
                return func(obj, obj1);
            }
            catch (Exception)
            {
                //log excetion
                throw;
            }
        }
    }
```