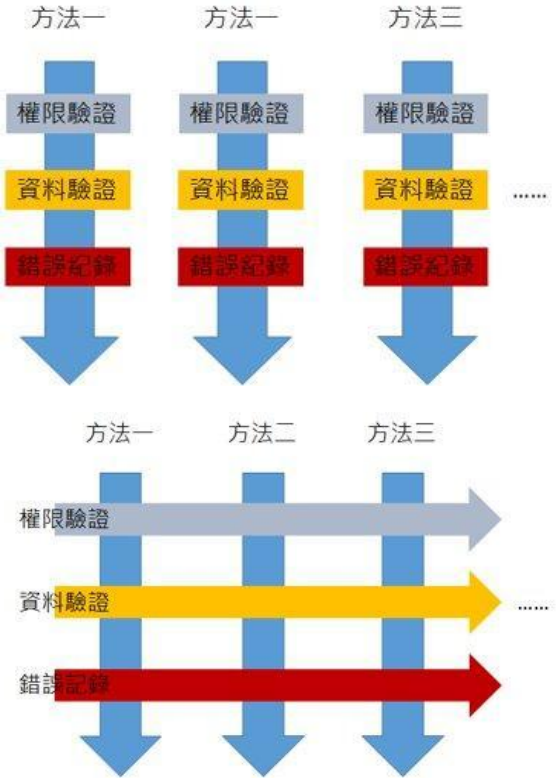
Dylan

AOP-剖面導向程式設計

將橫切關注點(Cross-cutting concerns)與業務主體進行進一步分離，以提高程式碼的模組化程度

平時我們撰寫程式時經常在主要邏輯前後加入身分驗證、日誌紀錄的程式碼，使每段商業邏輯中不斷夾雜且重複業務外的流程。

AOP框架可以將這些額外的流程切離，使你的程式碼的關注在商業邏輯中。



為什麼我們需要AOP?

- 這是一段排程程式碼

AOP框架 - AspectCore

- AOP & AspectCore 能做到什麼?
- Library AspectCore

1. 自訂攔截器，在進入Method前、離開Method後執行自訂邏輯
2. 攔截器內可以取得Method的參數、回傳值
3. 有多元的作法將攔截器套用到你想攔截的類別或方法

使用AspectCore將行為拆解

- 將Log與計時器做一個全域攔截器(Interceptor)
- 將AccessKey驗證機制做成一個特性攔截器(Attribute)

```
2 個參考
public class ScheduleLogInterceptor : AbstractInterceptor
{
    0 個參考
    public override async Task Invoke(AspectContext context, AspectDelegate next)
    {
        //執行方法前的操作
        var sw = new Stopwatch();
        sw.Start();

        var className = context.ImplementationMethod?.DeclaringType?.Name;
        var methodName = context.ServiceMethod.Name;
        var parameter = JsonConvert.SerializeObject(context.Parameters);
        Log.Info($"{className} {methodName} Schedule Start");
        Log.Info($"Parameter: {parameter}");

        await next(context);

        //執行方法後的操作
        var returnValue = $"{context.ReturnValue}";
        Log.Info($"Schedule End, Time: {sw.ElapsedMilliseconds}");
        Log.Info($"Return Value: {returnValue})");
        sw.Stop();
    }
}
```

Interceptor

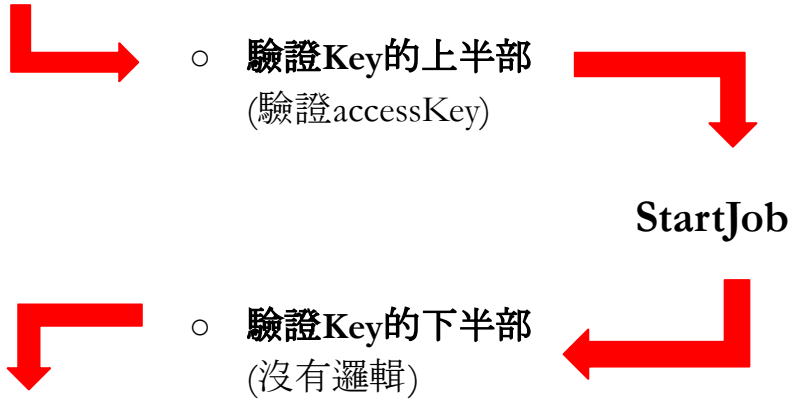
```
1 個參考
public class ScheduleValidInterceptorAttribute : AbstractInterceptorAttribute
{
    0 個參考
    public override async Task Invoke(AspectContext context, AspectDelegate next)
    {
        var accessKeyWithIndex = context.ServiceMethod
            .GetParameters()
            .Select((x, index) => new { Item = x, Index = index })
            .FirstOrDefault(x => x.Item.Name == "accessKey")!;

        var accessKey = context.Parameters[accessKeyWithIndex.Index].ToString();

        if (accessKey != "pass")
        {
            throw new Exception("Not Auth");
        }

        await next(context);
    }
}
```

Attribute

觸發流程

遇到await next 交給下一棒

- **計時Log的上半部**
  (計時開始、紀錄參數)
  - **驗證Key的上半部**
    (驗證accessKey)



  - **驗證Key的下半部**
    (沒有邏輯)
- **計時Log的下半部**
  (計時結束、紀錄返回 值)

設置全域攔截器需要代理的類別

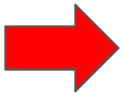
(特性攔截器不用設定代理，可以直接掛在Method上)

```
0 個參考
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    builder.Services.ConfigureDynamicProxy(config =>
    {
        config.Interceptors.AddTyped<ScheduleLogInterceptor>(Predicates.ForMethod("*Schedule"));
    });

    builder.Host.UseServiceProviderFactory(new DynamicProxyServiceProviderFactory());
```

成功瘦身

原有的功能都有保留下來，且分離出的邏輯可以輕易共用、開關

```
0 個參考
public void Schedule(string accessKey)
{
    var sw = new Stopwatch();
    Log.Info($"Schedule Start, Parameter = {accessKey}");

    var isPass = VerifyAccessKey(accessKey);

    if(!isPass)
    {
        throw new Exception("未授權");
    }

    StartJob();

    Log.Info($"Schedule End, ProcessTime = {sw.ElapsedMilliseconds}");
    sw.Stop();
}
```



```
[ScheduleValidInterceptor]
2 個參考
public void Schedule(string accessKey)
{
    StartJob();
}
```

實務上的應用

1. 老闆說最近網站很卡但不清楚原因，請你幫忙在所有的DB Query補上執行時間的Log。

   AddType<LogInterceptor>(Predicates.ForService("*Repository"));

2. 有個資料計算錯誤的issue找不到原因，你需要知道所有路徑上的Method參數和返回值。

3. 在不改動原有邏輯的情況下套用快取。