**定制化Audit.NET实现自定义AuditTarget**

**需求背景**

最近在项目上需要增加对用户操作进行审计日志记录的功能，调研了一圈，在.net core生态里，用的最多的是[Audit.NET](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Fthepirat000%2FAudit.NET" \t "_blank)。浏览完这个库的文档后，觉得大致能满足我们的诉求，于是建立一个控制台项目来先玩一玩。

但是我们还有额外的需求：

* 我们要记录的数据中包含了一些用户的敏感信息，这些内容是肯定不能记到审计日志里面的，所以得想个办法在写日志的时候把这些内容给去掉，这是这篇文章要解决的问题。
* 我们需要将日志数据发送到AWS的Simple Queue Service里面去，但是官方提供的一些预定义的DataProvider里没有这个功能，需要自己实现，这部分就不在这篇文章里说了，下次再写一篇。

**Audit.NET的基本使用方法**

**安装**

新建一个.net core console application，我的源代码在这里：[TryCustomAuditNet](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2FAsinta%2FTryCustomAuditNet" \t "_blank), 使用Nuget查找Audit.NET安装到项目中即可。

**配置**

这个库的官方文档已经有比较详细的配置项说明了，在这里我就不复述了，只记录一下基本配置。

static void Main(string[] args)

{

ConfigureAudit();

}

private static void ConfigureAudit()

{

Audit.Core.Configuration.Setup()

.UseFileLogProvider(config => config

.DirectoryBuilder(\_ => "./")

.FilenameBuilder(auditEvent => $"{auditEvent.EventType}\_{DateTime.Now.Ticks}.json"));

}

**使用**

**定义数据对象**

首先我们模拟一个需要被审计的数据对象Order，写一个方法用来修改其中一个属性：

public class Order

{

public Guid Id { get; set; }

public string CustomerName { get; set; }

public int TotalAmount { get; set; }

public DateTime OrderTime { get; set; }

public Order(Guid id, string customerName, int totalAmount, DateTime orderTime)

{

Id = id;

CustomerName = customerName;

TotalAmount = totalAmount;

OrderTime = orderTime;

}

public void UpdateOrderAmount(int newOrderAmount)

{

TotalAmount = newOrderAmount;

}

}

**业务逻辑中进行审计**

static void Main(string[] args)

{

ConfigureAudit();

var order = new Order(Guid.NewGuid(), "Jone Doe", 100, DateTime.UtcNow);

// 追踪order的审计

using (var scope = AuditScope.Create("Order::Update", () => order))

{

order.UpdateOrderAmount(200);

// optional

scope.Comment("this is a test for update order.");

}

}

**效果**

运行程序，在TryCustomAuditNet/bin/Debug/netcoreapp3.1目录下生成了一个审计日志文件Order::Update\_637408091235053310.json, 内容如下：

$ cat Order::Update\_637408091235053310.json

{

"EventType": "Order::Update",

"Environment": {

"UserName": "yu.li1",

"MachineName": "Yus-MacBook-Pro",

"DomainName": "Yus-MacBook-Pro",

"CallingMethodName": "TryCustomAuditNet.Program.Main()",

"AssemblyName": "TryCustomAuditNet, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null",

"Culture": ""

},

"Target": {

"Type": "Order",

"Old": {

"Id": "411b43b5-24be-4368-8978-2bf334e7ce8e",

"CustomerName": "Jone Doe",

"TotalAmount": 100,

"OrderTime": "2020-11-12T12:18:43.177177Z"

},

"New": {

"Id": "411b43b5-24be-4368-8978-2bf334e7ce8e",

"CustomerName": "Jone Doe",

"TotalAmount": 200,

"OrderTime": "2020-11-12T12:18:43.177177Z"

}

},

"Comments": [

"this is a test for update order."

],

"StartDate": "2020-11-12T12:18:43.212662Z",

"EndDate": "2020-11-12T12:18:43.498007Z",

"Duration": 285

}

可以看到在审计过程中，数据对象的TotalAmount值从100更新为了200，并且新增了一个Comments字段。

**问题**

我们的问题是，在审计日志中，不希望记录CustomerName这个字段的值，因为具体的人名被认为是显式的隐私数据，而这是不能直接记录到审计日志中的，怎么处理？

**解决方案**

一个比较简单粗暴的方法就是在需要记录审计日志的地方，将原始的数据对象经过映射之后传到AuditScope内部，但是这有几个问题：一是这样一来需要在程序中写大量不同的数据对象映射方法，不利于维护；二是我没有实验这种方式的开销有多大以及到底能不能准确实现我们的需求。所以我们去看看源码，然后整理一下思路。

**核心代码**

打开[Audit.NET](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Fthepirat000%2FAudit.NET" \t "_blank)的源代码，结合测试程序，我们定位到了几个关键的代码块：

**AuditScope.cs**

public partial class AuditScope : IAuditScope

{

private readonly AuditScopeOptions \_options;

#region Constructors

[MethodImpl(MethodImplOptions.NoInlining)]

internal AuditScope(AuditScopeOptions options)

{

\_options = options;

\_creationPolicy = options.CreationPolicy ?? Configuration.CreationPolicy;

\_dataProvider = options.DataProvider ?? Configuration.DataProvider;

\_targetGetter = options.TargetGetter;

// ... 省略中间代码

if (options.TargetGetter != null)

{

var targetValue = options.TargetGetter.Invoke();

\_event.Target = new AuditTarget

{

// IMPORTANT: 调用了AuditDataProvider中的Serialize方法来序列化数据对象

Old = \_dataProvider.Serialize(targetValue),

Type = targetValue?.GetType().GetFullTypeName() ?? "Object"

};

}

ProcessExtraFields(options.ExtraFields);

}

// ...省略其他代码

}

**AuditDataProvider.cs**

// IMPORTANT: AuditDataProvider这是一个抽象基类，我们可以通过继承AuditDataProvider实现自己的DataProvider。

public virtual object Serialize<T>(T value)

{

// IMPORTANT：重写这个方法，在重写中实现基于Attribute的数据对象字段过滤。

if (value == null)

{

return null;

}

return JToken.FromObject(value, JsonSerializer.Create(Configuration.JsonSettings));

}

**基本思路**

基本思路就是我们设法在需要记录的数据对象定义里，给需要或者不需要记录的属性加上自定义的Attribute，并且实现自己的DataProvider类重写Serialize方法，在序列化对象的时候根据这个特定的Attribute来过滤需要序列化的字段。

那么就搞起来。

**代码实现**

**添加自定义Attribue**

新建类UnAuditableAttribute，实现代码：

[AttributeUsage(AttributeTargets.Property)]

public class UnAuditableAttribute: Attribute

{

}

为我们不希望被审计的属性添加Attribute：

public class Order

{

public Guid Id { get; set; }

[UnAuditable]

public string CustomerName { get; set; }

// ...省略其他内容

}

**添加自定义DataProvider并重写关键方法**

为了简单，我们直接复制一份FileDataProvider类的内容到我们新建的CustomFileDataProvider类中：

public class CustomFileDataProvider: AuditDataProvider

{

public override object Serialize<T>(T value)

{

if (value == null)

{

return null;

}

// REGION START: 过滤属性

var jo = new JObject();

var serializer = JsonSerializer.Create(Configuration.JsonSettings);

foreach (PropertyInfo propInfo in value.GetType().GetProperties())

{

if (propInfo.CanRead)

{

object propVal = propInfo.GetValue(value, null);

var cutomAttribute = propInfo.GetCustomAttribute<UnAuditableAttribute>();

if (cutomAttribute == null)

{

// 被打上UnAuditableAttribute标记的属性，不加入序列化中。

jo.Add(propInfo.Name, JToken.FromObject(propVal, serializer));

}

}

}

// REGION END

return JToken.FromObject(jo, serializer);

}

public CustomFileDataProvider(Action<IFileLogProviderConfigurator> config)

{

// 为了在我们的测试工程中编译通过，需要自定义一个CustomFileDataProviderConfigurator类，实现照搬FileDataProviderConfigurator，只是将字段改为public的。

var fileConfig = new CustomFileDataProviderConfigurator();

if (config != null)

{

config.Invoke(fileConfig);

\_directoryPath = fileConfig.\_directoryPath;

\_directoryPathBuilder = fileConfig.\_directoryPathBuilder;

\_filenameBuilder = fileConfig.\_filenameBuilder;

\_filenamePrefix = fileConfig.\_filenamePrefix;

JsonSettings = fileConfig.\_jsonSettings;

}

}

// ...省略其他相同的内容

}

**修改配置**

最后我们修改一下最初的配置，使用我们自定义的DataProvider：

private static void ConfigureAudit()

{

Audit.Core.Configuration.Setup()

.UseCustomProvider(new CustomFileDataProvider(config => config

.DirectoryBuilder(\_ => "./")

.FilenameBuilder(auditEvent => $"{auditEvent.EventType}\_{DateTime.Now.Ticks}.json")

.JsonSettings(new JsonSerializerSettings

{

Formatting = Formatting.Indented,

ReferenceLoopHandling = ReferenceLoopHandling.Ignore,

NullValueHandling = NullValueHandling.Include

})));

}

**测试**

再运行一次程序，我们来看生成的审计文件：

$ cat Order::Update\_637408110805063180.json

{

"EventType": "Order::Update",

"Environment": {

"UserName": "yu.li1",

"MachineName": "Yus-MacBook-Pro",

"DomainName": "Yus-MacBook-Pro",

"CallingMethodName": "TryCustomAuditNet.Program.Main()",

"AssemblyName": "TryCustomAuditNet, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null",

"Culture": ""

},

"Target": {

"Type": "Order",

"Old": {

"Id": "f5c08f91-c0e4-4c25-b5ba-97edfb418346",

"TotalAmount": 100,

"OrderTime": "2020-11-12T12:51:19.081762Z"

},

"New": {

"Id": "f5c08f91-c0e4-4c25-b5ba-97edfb418346",

"TotalAmount": 200,

"OrderTime": "2020-11-12T12:51:19.081762Z"

}

},

"Comments": [

"this is a test for update order."

],

"StartDate": "2020-11-12T12:51:19.215596Z",

"EndDate": "2020-11-12T12:51:20.472729Z",

"Duration": 1257

}

注意到新的审计日志中已经不再包含CustomerName这个属性了，完美。

**总结**

[Audit.NET](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Fthepirat000%2FAudit.NET)这个框架还是非常强大的，这篇文章只探讨了其中非常小的一个特性点，当然基于本文的思路，我们还可以实现更复杂的审计日志数据对象过滤逻辑，可扩展性还是很好的。

如果不出意外，下一篇会写一写怎么实现一个SQSDataProvider类来实现将审计日志扔到AWS的Simple Queue Service中去的功能。

原文作者：[Asinta](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.asinta.cn%2F)  
原文链接：[https://www.asinta.cn/2020/11/12/Customize-Audit-Net-with-attibutes/](https://links.jianshu.com/go?to=https%3A%2F%2Fashy-stone-0d95aee00.azurestaticapps.net%2F2020%2F11%2F12%2FCustomize-Audit-Net-with-attibutes%2F)  
版权声明：本文采用[知识共享署名-非商业性使用 4.0 国际许可协议](https://links.jianshu.com/go?to=http%3A%2F%2Fcreativecommons.org%2Flicenses%2Fby-nc%2F4.0%2F)进行许可