PostSharp. OnMethodBoundary Aspect

Прочитать статью на сайте

Продолжаем подробный разговор о аспекте OnMethodBoundary. Данный аспект применяется, когда у вас есть доступ к исходным кодам метода и вы хотите отслеживать события начала вызова метода, конец вызова, исключение и успешность завершения. Все это делается переопределением методов *OnEntry*, *OnExit*, *OnException* и *OnSuccess* соответственно.

Применение аспекта OnMethodBoundary рассмотрим на примере задач о разрешении или запрещении тех или иных действий, логирование интересующих нас действий, генерация служебного кода. Так же с помощью данного аспекта можно декларативно указывать запускать ли методы в отдельном потоке или нет, но это рассмотрим подробно и в деталях в другой статье. Сегодня на повестке дня безопасность – это будет показано в одном приложении, исходный код которого можно свободно скачать и поиграться с ним.

Security

При создании более-менее сложных программных продуктов, часто требуется определять несколько ролей для работы с программой. В самом минимальном случае это администратор и пользователь. Для такого набора ролей можно как-то вывернутся по-простому, например, разграничить экраны для этих ролей, может даже каким-то образом сервисы. Но это не избавит от ошибок использования методов. При развитии программы могут появляться новые роли, менятся ответственность старых, и тогда придется отслеживать какой метод доступен для какой роли. В коде это должно превратиться в сущий ад и добавление новой роли или изменение ответственности будет не самой приятной задачей. К тому же вероятность ошибки все равно будет велика. Лучше определять роли доступа декларативно, таким образом будет легче управлять правами и сразу видно, что кому доступно. Легко вносить изменения и добавлять новые роли. Но обо всем по порядку.

Пусть у нас есть некий сервис зарплат, который конечно же должен быть разделен по ролям доступа. Обычный сотрудник может увидеть только свою зарплату, менеджер может увидеть зп всех своих сотрудников. Роль пользователя может храниться в контексте приложения и определяется при запуске приложения.

```
public class SalaryService {
        public decimal GetMySalary() {
            // some code
        }

        public decimal GetSalaryFor(int employeeId) {
            // some code
        }
    }
```

В самом простом случае, в основной сборке программы можем определить список возможных ролей и писать аспект основываясь на них.

Аспект будет на основе OnMethodBoundary и переопределим мы только метод **OnEntry**.

```
[Serializable]
    public class SecurityPermissions : OnMethodBoundaryAspect {
```

```
        public override void OnEntry(MethodExecutionEventArgs eventArgs) {
            base.OnEntry(eventArgs);
        }
    }
```

Пока у нас есть доступ только к самому методу и к экземпляру класса, к текущей роли пользователя. Роли, авторизированные для выполнения операции можно передавать через свойства создания атрибута.

В класс атрибута обявляем поле в котором будет массив доступных ролей:

```
public class SecurityPermissions : OnMethodBoundaryAspect {
        public SecurityRole[] AccessRoles { get; set; }

        public override void OnEntry(MethodExecutionEventArgs eventArgs) {… }
```

При применении атрибута, можно теперь будет указать роли, для которых вызов метода будет легитимным.

```
public class SalaryService {
        [SecurityPermissions(AccessRoles = new[] {SecurityRole.User,
SecurityRole.Manager})]
        public decimal GetMySalary() {
            // some code
        }

        [SecurityPermissions(AccessRoles = new[] {SecurityRole.Manager})]
        public decimal GetSalaryFor(int employeeId) {
            // some code
        }
    }
```

Реализация проверки в аспекте может выглядеть таким образом:

```
public override void OnEntry(MethodExecutionEventArgs eventArgs) {
            var accessRoles = new List<SecurityRole>(AccessRoles);
            if (!accessRoles.Contains(AppContext.Role)) {
                var exceptionText = string.Format("Method {0} is forbidden
for current user's role", eventArgs.Method.Name);
                throw new SecurityException(exceptionText);
            }

            base.OnEntry(eventArgs);
        }
```

В данном случае, если происходит попытка доступа к объекту который для роли не доступен – бросается исключение. Такое может произойти при ошибке применения сервиса.

Далее аспект надо будет доработать в плане возможностей наложения, применения, как в прошлых статьях. И можно пользоваться. Вообще тема безопасности огромна, и только таким решением не ограничишься. Но это как всегда тема отдельной статьи =)

Исходные коды как всегда доступны в полном объеме. Не забудьте только скачать и установить PostSharp. «Happy Postsharping!» – как говорит основатель проекта Gael Fraiteur, и…

Hard’n’heavy!