Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Аспектно-ориентированное программирование

© Марченко Антон Александрович marchenko@it.kfu.ru Казань, 2020 г.

Модульность

- Это круто!
- Классы, методы, библиотеки классов/сборки, микросервисы
  - «Вынеси запрос в отдельный метод»
  - «Вынеси работу с пользователями в отдельный класс»
  - «Вынеси все компоненты работы с БД в отдельный проект библиотеки классов»

Но…

- Вынеси о отдельный метод проверку авторизации
- Вынеси в отдельный модуль логирование
- Вынеси в отдельный класс обработку исключений

Мотивация

Код на Java взят из статьи
https://habrahabr.ru/post/114649/

```
public BookDTO getBook(Integer bookId) {
  BookDTO book = bookDAO.readBook(bookId);
  return book;
}
```

```
public BookDTO getBook(Integer bookId) throws
ServiceException, AuthException {
  if (!SecurityContext.getUser().hasRight("GetBook"))
    throw new AuthException("Permission Denied");

  LOG.debug("Call method getBook with id " + bookId);
  BookDTO book = null;
  String cacheKey = "getBook:" + bookId;

  try {
    if (cache.contains(cacheKey)) {
      book = (BookDTO) cache.get(cacheKey);
    } else {
      book = bookDAO.readBook(bookId);
      cache.put(cacheKey, book);
    }
  } catch(SQLException e) {
    throw new ServiceException(e);
  }

  LOG.debug("Book info is: " + book.toString());
  return book;
}
```

Что плохого?

```
public BookDTO getBook(Integer bookId)
            throws ServiceException, AuthException {
  if (!SecurityContext.getUser().hasRight("GetBook"))
    throw new AuthException("Permission Denied");

  LOG.debug("Call method getBook with id " + bookId);
  BookDTO book = null;
  String cacheKey = "getBook:" + bookId;

  try {
    if (cache.contains(cacheKey)) {
      book = (BookDTO) cache.get(cacheKey);
    } else {
      book = bookDAO.readBook(bookId);
      cache.put(cacheKey, book);
    }
  } catch(SQLException e) {
    throw new ServiceException(e);
  }

  LOG.debug("Book info is: " + book.toString());
  return book;
}
```

Права доступа

Кэширование

Исключения

Логирование

Сквозной функционал

- *Scattered, Tangled (рассеянный, спутанный)*
- Функционал, который нельзя просто так взять и выделить в отдельные модули
  - Обработка исключений
  - Логирование
  - Трассировка
  - Проверка прав доступа
  - Кэширование
  - …

Boilerplate

- Блоки кода, повторяемые в нескольких местах с минимальными изменениями
- Когда пишем много кода для решения простой задачи
- Усложняет код, отвлекая от бизнес логики

Задача

- Переместить сквозной вспомогательный код в подходящее место
- Сделать его переиспользуемым

Cross-cutting concerns

- Разделение основной и второстепенной функциональности
- Core concern – основная функциональность, например, бизнес-логика
- Cross-cut concern – сквозная функциональность

Парадигма

- Подход к написанию программ
- Совокупность идей и понятий
- В основе единица программы:
  - инструкция (императивное программирование)
  - функция (функциональное программирование)
  - прототип (прототипное программирование)
  - объект (ООП)
  - факт (логическое программирование)
  - …

Нам нужен подход, в основе которого лежат модули, инкапсулирующие сквозную функциональность (аспекты) –> Аспектно-ориентированное программирование (АОР)

АОП и ООП

- АОП не замещает ООП, а дополняет
- Позволяет инкапсулировать то, что нельзя выделить в отдельный класс

Как готовить АОР?

- Вызывать сквозной функционал до, после, вместо основного

Схема работы:

- Переносим сквозной функционал в отдельный модуль - аспект
- Советуем как он должен вызываться и в каких точках подключения
- Получаем чистый код

Определения: Аспект и совет

- Aspect – инкапсулированная сквозная функциональность, нацеленная на решение конкретной задачи
- Advice – код функциональности аспекта, который будет применяться для объектов

Определения: Соединение и срез

- Join point – точка, определяющая где будет применяться совет (вызов метода, создание объекта, обращение к полю/переменной)
- Pointcut – срез, совокупность точек соединения, определяющая соответствие точки соединения совету
  - все get-методы свойств;
  - все методы, содержащие параметр string

Определения: цель и прикрепление

- Target – объект, к которому применяется сквозная функциональность аспекта
- Weaving – связывание объектов с аспектами (runtime или compile time)
- Introduction – изменение структуры класса или/и иерархии наследования для добавления функциональности аспекта

Подходы к реализации AOP

Weaving – переплетение, сборка модулей основной и сквозной функциональности

- Переплетение во время компиляции
- Переплетение при линковке – после генерации CIL кода
- Переплетение во время исполнения

https://www.c-sharpcorner.com/UploadFile/shivprasadk/aspect-oriented-programming-in-C-Sharp-net-part-i/

Реализации AOP в C#

- https://ayende.com/blog/2615/7-approaches-for-aop-in-net
- Язык и компилятор для генерации сборок .NET
- Изменение поведения сборок при исполнении
- Ставить между клиентом и целевым объектом заместителя (прокси), добавляющего сквозной функционал к вызовам

Первые два подхода

- Мощные инструменты с высокой производительностью
- Которые чрезвычайно сложно реализовывать
- И так же тяжело отлаживать

Подход с заместителем

- Во время исполнения нужно подменять целевой объект заместителем
- Если нужно добавлять функционал к конструкторам, нужно делегировать создание экземпляров фабрике

Использование: Attributes vs Interception

- Атрибуты
  - Просто использовать
  - Сложно модифицировать
  - Увеличивается связанность
  - Код обрастает атрибутами
- Перехват вызовов
  - Просто модифицировать
  - Сложнее использовать
  - В целевом коде ничего не указано про возможное прикрепление (код чище, но сложнее разобраться в работе сквозного функционала)

Реализация AOP на основе DI

- DI имеет встроенные возможности по созданию экземпляров, управлению зависимостями и позволяет реализовать перехват методов интерфейса
- Реализация AOP на DI – одна из самых удобных и жизнеспособных

AOP vs DI

- DI – для ослабления зависимостей между компонентами через связи с интерфейсами-контрактами, а не реализациями
- Контейнер – фабрика, отвечает за связывание объектов, выбор реализаций

AOP vs DI

- AOP – для чистоты кода
  - борьба с boilerplate, устранение дублирования сквозного кода
- AOP решает различные проблемы

AOP vs DI

Сходства:

- Добиваются слабой связанности
- Обеспечивают лучшее разделение функционала
- Разгружают основной код

Отличия:

- DI применяется для управления зависимостями
- В AOP целевой код не обязательно зависит от поведения совета

Проект Castle.DynamicProxy

- Модифицируемый метод должен быть доступным и виртуальным
- Модификация с помощью перехватчиков - классов, реализующих интерфейс IInterceptor, содержащий метод
  void Intercept (IInvocation invocation)

http://www.castleproject.org/projects/dynamicproxy/

Пример

```
public class TransactionScoper : IInterceptor
{
    public void Intercept(IInvocation invocation)
    {
        using (var tr = new TransactionScope())
        {
            invocation.Proceed();
            tr.Complete();
        }
    }
}
```

```
var generator = new ProxyGenerator();
var foo = new Foo();
var fooInterfaceProxyWithCallLogerInterceptor
    = generator.CreateInterfaceProxyWithTarget
      (foo, TransactionScoper);
```

```
var builder = new ContainerBuilder();
builder.Register(c => new TransactionScoper());
builder.RegisterType<Foo>()
    .As<IFoo>()
    .InterceptedBy(typeof(TransactionScoper));

var container = builder.Build();
var willBeIntercepted = container.Resolve<IFoo>();
```

https://habrahabr.ru/post/305360/

Еще пример

```
public class CallLogger : IInterceptor
{
    public readonly TextWriter Output;
    public CallLogger(TextWriter output)
    {
        Output = output;
    }
    public void Intercept(IInvocation invocation)
    {
        Output.WriteLine("Calling method {0} with parameters {1}.",
            invocation.Method.Name,
            string.Join(", ", invocation.Arguments
                                .Select(a => (a ?? "").ToString()).ToArray()));
        invocation.Proceed();
        Output.WriteLine("Done: result was {0}.", invocation.ReturnValue);
    }
}
```

Несколько перехватчиков

```
var fooInterfaceProxyWith2Interceptors =
generator.CreateInterfaceProxyWithTarget
(Foo, CallLogger, ErrorHandler);
```

Плюсы и минусы контейнерного AOP

+Чистая бизнес-логика

+Не нужно вручную выстраивать иерархии (как с «декоратором»)

-Зависимость от IoC контейнера

-Накладные расходы на proxy и перехват

-Нужно проектировать перехватчики

Boilerplate в ASP.NET Core

Общего характера

- Валидация
- Логирование
- Обработка исключений

Специфичные

- DI
- Repository, Unit Of Work
- Авторизация
- Локализация
- Mapping

Authorize, OutputCache

- Есть стандартные инструменты – система фильтрации:

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-3.1

ASP.NET Boilerplate

- Open source фреймворк, в котором аккумулированы решения типовых задач

https://aspnetboilerplate.com

https://github.com/aspnetboilerplate

AOP в ASP.NET Boilerplate

- Использует IoC контейнер Castle Windsor для перехвата

Что есть во фреймворке?

- Данные
  Репозитории, Unit Of Work
- Приложение
  Авторизация, валидация, логирование
- Общее

IoC контейнер, исключения, локализация

AOP. Как ещё?

- Autofac - использует DynamicProxy
- PostSharp

https://www.postsharp.net/

https://medium.com/AOP_PostSHarp

- Mono.Cecil

https://www.mono-project.com/docs/tools+libraries/libraries/Mono.Cecil/

Ссылки

1. https://habrahabr.ru/post/114649/
2. https://habrahabr.ru/post/305360/
3. https://habrahabr.ru/post/199378/
4. https://www.c-sharpcorner.com/uploadfile/shivprasadk/aspect-oriented-programming-in-c-sharp-net-part-i/
5. https://aspnetboilerplate.com/
6. https://github.com/aspnetboilerplate/aspnetboilerplate