ЛЕКЦИЯ

Многоуровневая (многослойная) архитектура

Лектор Крамар Ю.М.

Содержание

1. Многоуровневая (многослойная) архитектура
2. Внедрение зависимостей (DI)

МНОГОУРОВНЕВАЯ (МНОГОСЛОЙНАЯ) АРХИТЕКТУРА

Многоуровневая архитектура

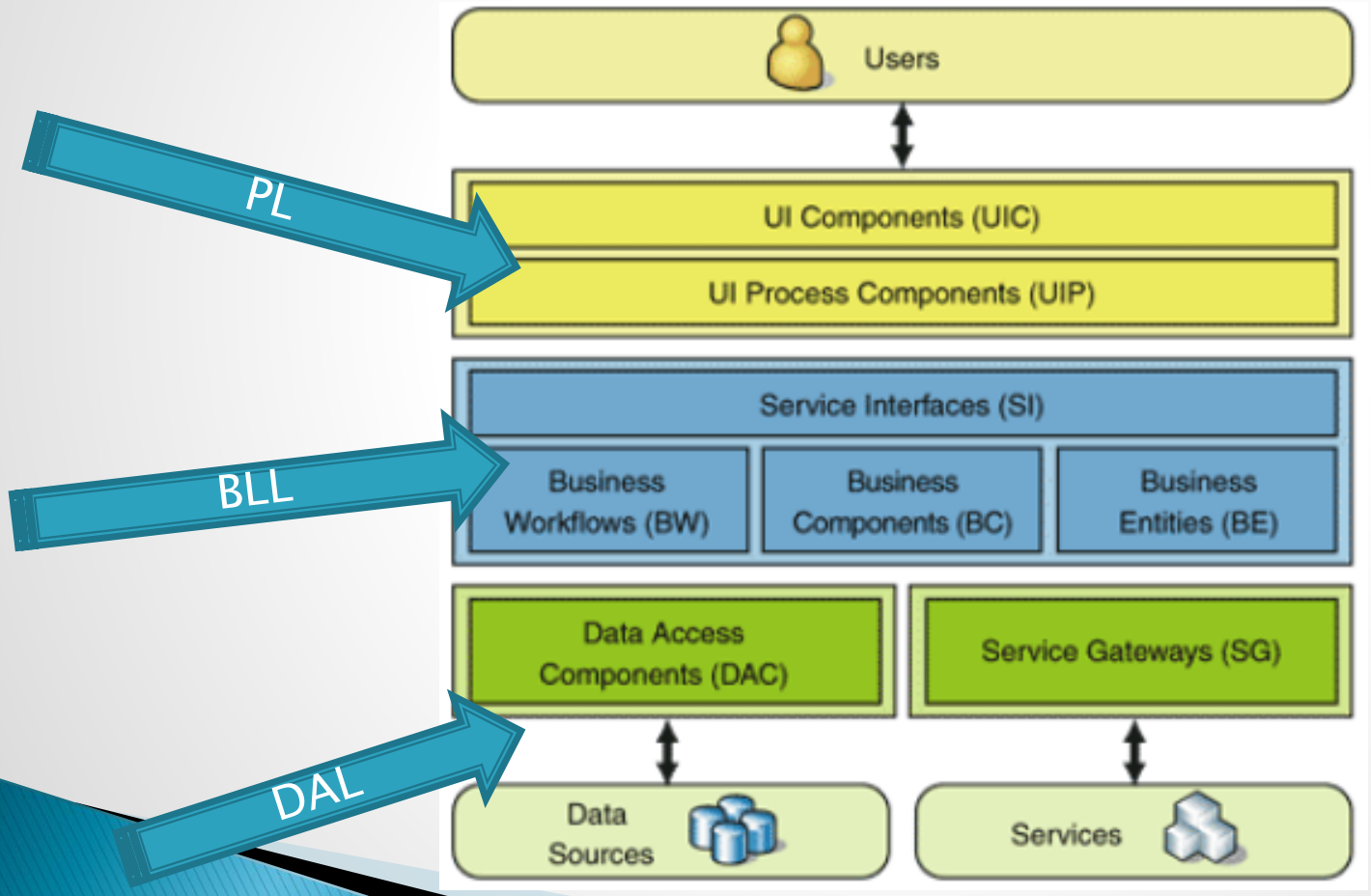
Многоуровневая архитектура является одной из архитектурных парадигм разработки ПО, при которой функциональные области приложения разделяются на слои.

При разработке архитектуры системы нужно определять каждый слой так, чтобы его можно было легко заменить на аналогичный или использовать повторно в другом приложении.

Каждый слой, расположенный выше, обладает информацией о том, как обратиться к слою, расположенному непосредственно под ним. И ни в коем случае не наоборот!!!

Многоуровневая архитектура

https://msdn.microsoft.com/en-us/library/ff648105.aspx



Многоуровневая архитектура

Presentation layer – уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя

Business layer – уровень, содержащий набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализующий всю необходимую логику приложения и передающий уровню представления результат обработки данных из БД.

Data Access layer – хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных Entity Framework. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.

Многоуровневая архитектура

Свойства уровней:

- Крайние уровни не могут взаимодействовать между собой, то есть уровень представления (применительно к ASP.NET MVC, контроллеры) не могут напрямую обращаться к базе данных и даже к уровню доступа к данным, а только через уровень бизнес-логики.
- Уровень доступа к данным не зависит от других уровней, уровень бизнес-логики зависит от уровня доступа к данным, а уровень представления – от уровня бизнес-логики.
- Компоненты, как правило, должны быть слабосвязанными (loose coupling), поэтому для многоуровневых приложений важным является применение инверсии управления (IoC) посредством внедрения зависимостей (DI).
- Один уровень может представлять несколько проектов, главное, чтобы его функционал представлял единое логическое звено.

Многоуровневая архитектура

- каждый слой занимается конкретной задачей.
- логика разных слоёв не повторяется и не пересекается
- способ обращения к нижестоящему слою чётко определён
- способ поставки информации вышестоящему слою чётко определён
- слои слабо связаны
- слои расположены вертикально, хотя есть сквозная функциональность, которая может пронизывать пирамиду сверху вниз
- слои могут размещаться физически на одном компьютере (в пределах одного уровня), а могут быть на разных машинах, например, в распределённых приложениях.
- логика разных слоёв инкапсулирована, соответственно, разным слоям не нужно делать никаких предположений о том, как реализован код других слоёв приложения.
- возможность использования в других сценариях (за счёт слабого связывания и чётко определённой задачи).

Многоуровневая архитектура

Один из SOLID-принципов – **принцип инверсии зависимостей** (**Dependency inversion principle,** DIP):

Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракции.

Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Как реализовать: заменить композицию агрегацией. То есть вместо создания зависимостей напрямую, класс должен требовать их у более высокого уровня через аргументы метода или конструктора. При этом зависимость должна передаваться не в виде экземпляров конкретных классов, а в виде интерфейсов или абстрактных классов

DI (DEPENDENCY INJECTION) – ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ

Многоуровневая архитектура

Инверсия управления (Inversion of Control, IoC) — важный принцип объектно-ориентированного программирования, используемый для уменьшения сцепления в компьютерных программах. Он представляет набор рекомендаций для написания слабо связанного кода. Суть его в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

Примеры реализации IoC:

- Шаблон "Фабрика" (Factory pattern)
- Service locator
- Внедрение зависимости (Dependency injection)

Многоуровневая архитектура

Наиболее распространенной реализацией IoC в применении к управлению зависимостями является **внедрение зависимостей (Dependency injection).**

Существует **три типа DI**:

- через конструктор (Constructor Injection)
- через метод (Method Injection)
- через свойство (Property Injection)

Через конструктор передаются обязательные зависимости класса, без которых работа класса невозможна Через метод передаются зависимости, которые нужны лишь одному методу, а не всем методам класса и должна быть возможность менять от вызова к вызову. Через свойства должны устанавливаться лишь необязательные зависимости (обычно, инфраструктурные), для которых существует значение по умолчанию.

Многоуровневая архитектура

```
class ReportProcessor
{
    private readonly IReportSender _reportSender;

    // Constuctor Injection: передача обязательной зависимости
    public ReportProcessor(IReportSender reportSender)
    {
        _reportSender = reportSender;
        Logger = LogManager.DefaultLogger;
    }

    // Method Injection: передача обязательных зависимостей метода
    public void SendReport(Report report, IReportFormatter formatter)
    {
        Logger.Info("Sending report...");
        var formattedReport = formatter.Format(report);
        _reportSender.SendReport(formattedReport);
        Logger.Info("Report has been sent");
    }

    // Property Injection: установка необязательных "инфраструктурных" зависимостей
    public ILogger Logger { get; set; }
}
```

Многоуровневая архитектура

Для реализации DI используются IoC-контейнеры.

IoC-контейнер — это инструмент (библиотека, фреймворк), который позволит упростить и автоматизировать написание кода с использованием DI-подхода (Ninject, Autofac, Unity и др.)

```
using Ninject.Modules;
using Ninject;

var kernel = new StandardKernel();
kernel.Bind<IReportSender>().To<MockReportSender>();
var mailSender = kernel.Get<IReportSender>();
var ReportProcessor = new ReportProcessor(mailSender);

// or

public class Bindings : NinjectModule
{
    public override void Load()
    {
        Bind<IReportSender>().To<MockReportSender>();
    }
}

IKernel ninjectKernel = new StandardKernel(new Bindings());
ReportProcessor reportProcessor = ninjectKernel.Get<ReportProcessor>();
```

Code

Документация по Ninject: https://github.com/ninject/Ninject/wiki

Ресурсы

1. https://msdn.microsoft.com/ru-ru/library/bb399567%28v=vs.110%29.aspx
2. https://msdn.microsoft.com/en-us/library/ff648105.aspx
3. http://metanit.com/sharp/entityframework/1.1.php
4. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648105(v=pandp.10)?redirectedfrom=MSDN
5. https://metanit.com/sharp/mvc5/23.5.php
6. https://metanit.com/sharp/mvc5/23.6.php
7. https://metanit.com/sharp/mvc5/23.1.php