# Врезка: DI vs. DIP vs. IoC

Существует три схожих понятия, связанных с передачей зависимостей, в каждом из которых есть слово "инверсия" (inversion) или "зависимость" (dependency):

* IoC – Inversion of Control (Инверсия управления)
* DI – Dependency Injection (Внедрение зависимостей)
* DIP – Dependency Inversion Principle (Принцип инверсии зависимостей)

Подливает масло в огонь рассогласованность использования этих терминов. Так, например, контейнеры иногда называют DI-контейнерами, а иногда IoC-контейнерами. Большинство разработчиков не различает DI и DIP, хотя за каждой из этих аббревиатур скрываются разные понятия.

## Inversion of Control (IoC)

Инверсия управления (IoC, Inversion of Control) – это достаточно общее понятие, которое отличает библиотеку от фреймворка. Классическая модель подразумевает, что вызывающий код контролирует внешнее окружение, время и порядок вызова библиотечных методов. Однако в случае фреймворка обязанности меняются местами: фреймворк предоставляет некоторые точки расширения, через которые он вызывает определенные методы пользовательского кода.

Простой метод обратного вызова или любая другая форма паттерна Наблюдатель является примером инверсии управления. Зная значение понятия IoC становится ясно, что такое понятие как IoC-контейнер лишено смысла, если только данный «контейнер» не предназначен для упрощения создания фрейморков.

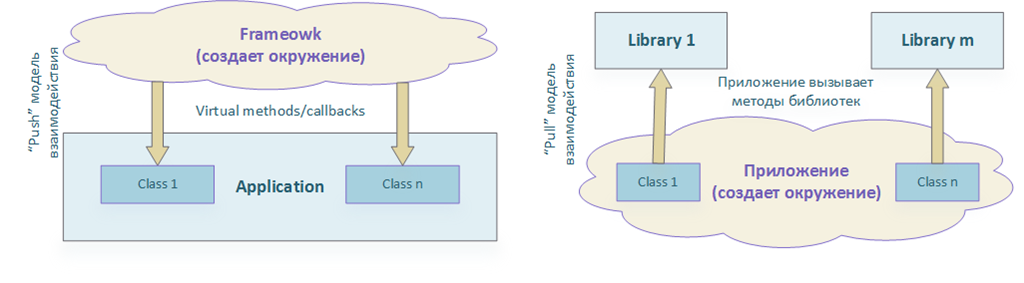


Рисунок 1 – Разница между фреймворком и библиотекой

## Dependency Injection (DI)

Внедрение зависимостей (DI, Dependency Injection) – это механизм передачи классу его зависимостей. Существует несколько конкретных видов или паттернов внедрения зависимостей: внедрение зависимости через конструктор (Constructor Injection), через метод (Method Injection) и через свойство (Property Injection).

Ниже приведены примеры использования каждого вида внедрения зависимостей (листинг 1):

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;}  
}

Листинг 1 – Виды внедрения зависимостей

Разные виды внедрения зависимостей предназначены для решения разных задач. Через конструктор передаются обязательные зависимости класса, без которых работа класса невозможна (в нашем примере, IReportSender - обязательная зависимость класса ReportProcessor). Через метод передаются зависимости, которые нужны лишь одному методу, а не всем методам класса (IReportFormatter необходим только методу отправки отчета, а не классу ReportProcessor целиком). Через свойства должны устанавливаться лишь необязательные зависимости (обычно, инфраструктурные), для которых существует значение по умолчанию (свойство Logger содержит разумное значение по умолчанию, но может быть заменено позднее).

Очень важно понимать, что DI-паттерны не говорят о том, как должна выглядеть зависимость, к какому уровню она относится, сколько их должно быть и т.п. Это лишь инструмент передачи зависимостей от одного класса другому. Ни больше, ни меньше!

## Dependency Inversion Principle (DIP)

Принцип инверсии зависимости говорит о том, какого вида зависимости необходимо передавать классу извне, а какие зависимости класс должен создавать самостоятельно. Важно, чтобы зависимости класса были понятны и важны вызывающему коду. Зависимости класса должны располагаться на текущем или более высоком уровне абстракции. Другими словами, не любой класс, который требует интерфейс в конструкторе, следует принципу инверсии зависимостей (листинг 2):

class ReportProcessor  
{  
 private readonly ISocket \_socket;  
 public ReportProcessor(ISocket socket)  
 {  
 \_socket = socket;  
 }  
  
 public void SendReport(Report report, IStringBuilder stringBuilder)  
 {  
 stringBuilder.AppendFormat(CreateHeader(report));  
 stringBuilder.AppendFormat(CreateBody(report));  
 stringBuilder.AppendFormat(CreateFooter(report));  
 \_socket.Connect();  
 \_socket.Send(ConvertToByteArray(stringBuilder));  
 }  
}

Листинг 2 – Пример нарушения DIP при наличии DI

Класс ReportProcessor все еще принимает "абстракцию" в аргументах конструктора - ISocket, но эта "абстракция" находится на несколько уровней ниже уровня формирования и отправки отчетов. Аналогично дела обстоят и с аргументом метода SendReport: "абстракция" IStringBuilder не соответствует принципу инверсии зависимостей, поскольку оперирует более низкоуровневыми понятиями, чем требуется. На этом уровне нужно оперировать не строками, а отчетами.

В результате, в данном примере используется внедрение зависимостей (DI), но данный код не следует принципу инверсии зависимостей (DIP).

Подведем итоги.

Инверсия управления (IoC) говорит об изменении потока исполнения, присуща фреймворкам и функциям обратного вызова и не имеет никакого отношения к управлению зависимостями. Передача зависимостей (DI) - это инструмент передачи классу его зависимости через конструктор, метод или свойство. Принцип инверсии зависимостей (DIP) - это принцип проектирования, который говорит, что классы должны зависеть от высокоуровневых абстракций.

## Дополнительные ссылки

* [DI Паттерны: Constructor Injection](http://sergeyteplyakov.blogspot.com/2012/12/di-constructor-injection.html))
* [DI Паттерны: Method Injection](http://sergeyteplyakov.blogspot.com/2013/02/di-method-injection.html)
* [DI Паттерны: Property Injection](http://sergeyteplyakov.blogspot.com/2013/01/di-property-injection.html)
* [DIP in the Wild](http://martinfowler.com/articles/dipInTheWild.html)
* [Inversion of Control Containers and the Dependency Injection pattern](http://martinfowler.com/articles/injection.html)
* [Фрейморки, библиотеки и зависимости](http://sergeyteplyakov.blogspot.com/2012/10/blog-post_26.html)