1. **Поясните понятие «внедрение зависимости» (DI).**

программный механизм, позволяющий в автоматическом режиме создавать программный объект, с заданными жизненным циклом (задаются события инстансирования и разрушения объекта), способом применения (в качестве параметра метода или конструктора, свойства или поля объекта) и областью действия.

1. **Для чего применяется DI?**

повторное применение кода, упрощает внесение изменений, упрощает тестирование

позволяет создавать слабосвязанные компоненты

1. **Что такое Ninject?**

Ninject - это мощный и одновременно легкий в использовании IoC контейнер.

IoC - Inversion of Control, принцип программирования, позволяющий снизить зависимость между компонентами программ; DI – один из способов реализации IoC.

1. **Поясните принцип подключения и конфигурации Ninject в ASP.NET-приложении.**

Чтобы управлять зависимостями через Ninject, вначале надо создать объект Ninject.IKernel с помощью встроенной реализации этого интерфейса - класса StandardKernel:

|  |  |
| --- | --- |
| 1 | IKernel ninjectKernel = new StandardKernel(); |

Далее нужно установить отношения между интерфейсами и их реализациями:

|  |  |
| --- | --- |
| 1 | ninjectKernel.Bind<IRepository>().To<BookRepository>(); |

Данное выражение указывает, что объекты IRepository должны будут рассматриваться как BookRepository.

И в конце создается объект интерфейса через метод Get:

|  |  |
| --- | --- |
| 1 | repo = ninjectKernel.Get<IRepository>(); |

Поскольку выше мы установили сопоставление между IRepository и BookRepository, то метод ninjectKernel.Get<IRepository>() будет создавать экземпляр класса BookRepository.

**Глобальная регистрация завимостей**

Зарегистрируем зависимости глобально для всех контроллеров и для этого создадим в проекте новую папку Util и поместим в нее новый класс NinjectRegistrations:

Класс NinjectRegistrations наследуется от класса NinjectModule и фактически представляет модуль Ninject. Он переопределяет метод Load(), который вызывается при загрузке модуля. И с помощью вызова Bind<IRepository>().To<BookRepository>(); собственно устанавливается сопоставление между интерфейсом-зависимостью и конкретным классом этого интерфейса.

И в конце необходимо инициализировать сопоставление зависимостей происходило при запуске приложения. Для этого перейдем к файлу **Global.asax.cs**, который запускается при старте приложения.

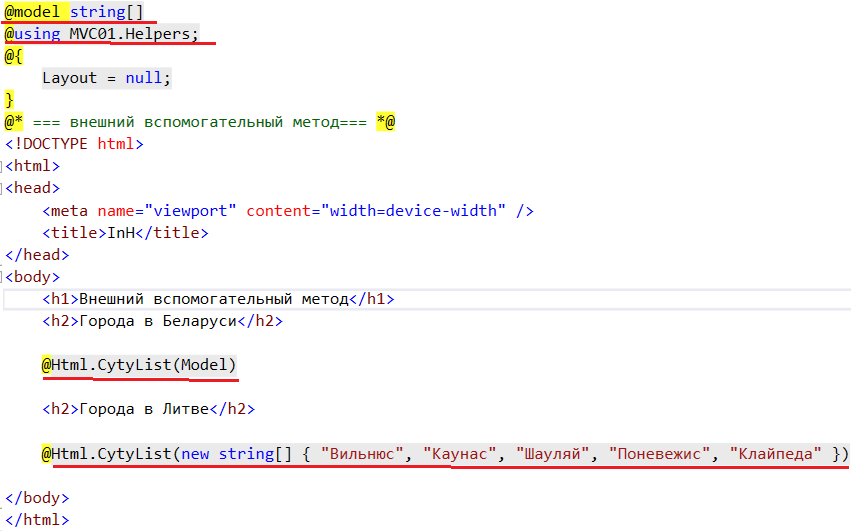
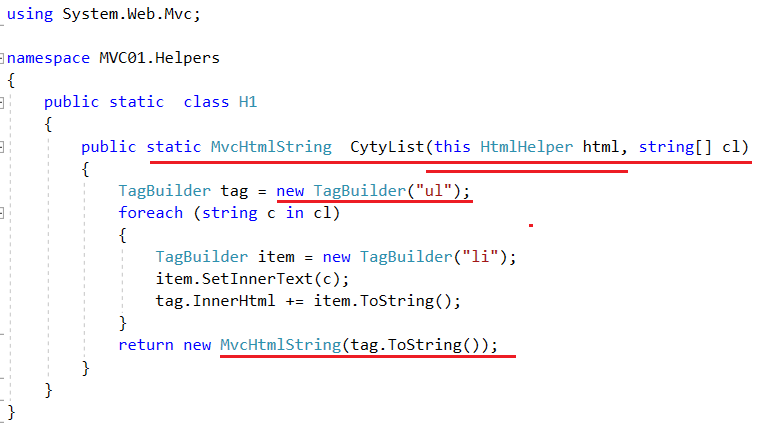
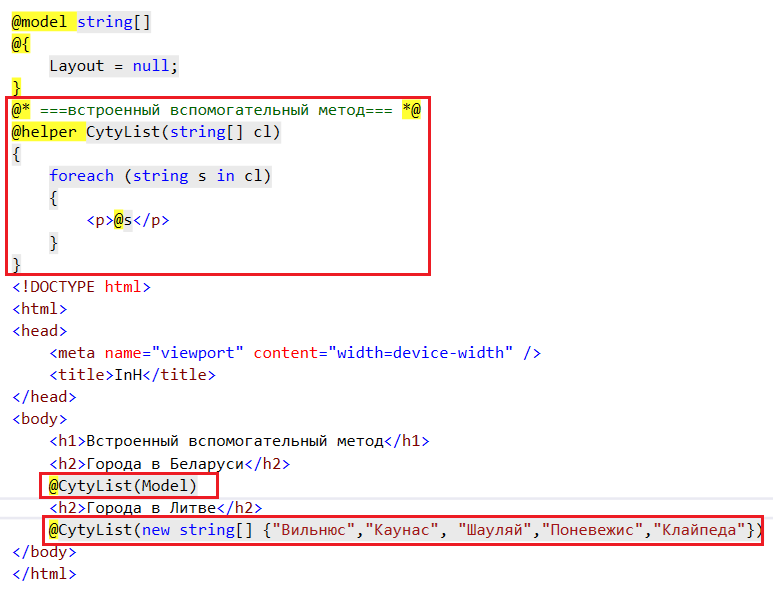
1. **Каким образом может быть выполнено DI в ASP.NET-приложении.**

Рассмотрим поэтапно, как происходит в данном случае внедрение зависимостей:

1. Фреймворк MVC получает запрос и обращается к контроллеру HomeController
2. Фреймворк MVC обращается к классу сопоставления зависимостей (в данном случае класс NinjectDependencyResolver), чтобы тот создал новый объект HomeController, передавая параметр Type в метод GetService (в класс NinjectDependencyResolver)
3. Сопоставитель зависимостей вызывает инфраструктуру Ninject для создания нового объекта HomeController, передавая тип создаваемого объекта в метод TryGet
4. Ninject смотрит на конструктор HomeController и видит, что там используется зависимость от интерфейса IRepository, для которого он устанавливает сопоставление с конкретной реализацией
5. Ninject создает экземпляр класса BookRepository и затем использует его для создания контроллера HomeController
6. Ninject передает созданный объект HomeController сопоставителю зависимостей, который, в свою очередь, передает его фреймворку MVC. И далее происходит обработка запроса.
7. **Назовите четыре вида DI, которые позволяет сделать Ninject в ASP.NET-приложении и поясните их особенности.**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| *InTransientScope()* | Аналогичен отсутствию указания области действия и создает новый объект для каждой распознаваемой зависимости |
| *InSingletonScope()* | Создает одиночный экземпляр, который разделяется по всему приложению. Ядро Ninject будет создавать экземпляр, если используется метод InSingletonScope(), или же экземпляр можно предоставить посредством метода ToConstant() |
| *InThreadScope()* | Создает одиночный экземпляр, который используется для распознавания зависимостей объектов, запрашиваемых в одном потоке |
| *InRequestScope()* | Создает одиночный экземпляр, который используется для распознавания зависимостей объектов, запрашиваемых в одном HTTP-запросе |

1. Поясните понятие «хэлпер».
2. Объясните последовательность разработки внутренних и внешних хэлперов.



1. Перечислите стандартные хэлперы входящие в состав ASP.NET.